AMSTRAD

GUIDE DU BASIC ET DE L'AMSDOS
CPC 464 / 664 / 6128

JEAN-LOUIS GRECO / MICHEL LAURENT



AMSTRAD
GUIDE DU BASIC ET DE L'AMSDOS

DANS LA MÊME COLLECTION

Amstrad jeux d'action, P. Monsaut
Amstrad premiers programmes, R. Zaks
Amstrad 56 programmes, S.R. Trost
Amstrad exploré, J. Braga
Amstrad programmation en assembleur, G. Fagot-Barraly
Amstrad guide du graphisme, J. Wynford
Amstrad CP/M 2.2, A. d'Hardancourt
Amstrad astrologie, numérologie, biorythmes, P. Bourgault
Amstrad graphisme en trois dimensions, T. Lachant-Robert
Amstrad Multiplan, Amstrad
Amstrad CP/M plus, A. d'Hardancourt
Amstrad Astrocalc, G. Blanc/P. Destrebecq
Amstrad gagnez aux courses, J.-C. Despoine
Amstrad créer de nouvelles instructions, J.-C. Despoine
Amstrad Locoscript, B. Le Dû
Amstrad Logo, A. d'Hardancourt *(à paraître)*
Amstrad mise au point des programmes BASIC, C. Vivier/J. Jacob *(à paraître)*
Amstrad programmes en langage machine, S. Webb *(à paraître)*
Amstrad guide du DOS, Amstrad *(à paraître)*
Amstrad introduction à la programmation en assembleur du Z80, A. d'Hardancourt *(à paraître)*
Amstrad systèmes d'exploitation, Amstrad *(à paraître)*
Amstrad programmes scientifiques, Y. Muggianu/M. Lamarche/P.-M. Beaufils *(à paraître)*
Amstrad routines en assembleur, J.-C. Despoine *(à paraître)*
Amstrad jeux en assembleur, E. Ravis *(à paraître)*
Amstrad mieux programmer en assembleur, T. Lachant-Robert *(à paraître)*

J.-L. GRÉCO/M. LAURENT

AMSTRAD
GUIDE DU BASIC ET DE L'AMSDOS



Paris • Berkeley • Düsseldorf

SYBEX est indépendant de tout constructeur.

Tous les efforts ont été faits pour fournir dans ce livre une information complète et exacte. Néanmoins, Sybex n'assume de responsabilités, ni pour son utilisation, ni pour les contrefaçons de brevets ou atteintes aux droits de tierces personnes qui pourraient résulter de cette utilisation.

ISBN 2-7361-0159-0

S O M M A I R E

1
INTRODUCTION

2
PROGRAMMER EN BASIC SUR AMSTRAD

3
INSTRUCTIONS, COMMANDES ET FONCTIONS BASIC DE L'AMSTRAD CPC 464, CPC 664 ET CPC 6128. COMMANDES AMSDOS

1

INTRODUCTION

Le développement de la micro-informatique familiale a été marqué, en 1984, par l'apparition sur le marché français des premiers ordinateurs Amstrad. Leur succès croissant tient probablement à la qualité du produit, mais aussi au fait qu'Amstrad offre un système *clés en main* pour un prix des plus abordables. Fini le temps où, additionnant le prix de l'unité centrale et celui des unités périphériques indispensables que l'on se réservait pour plus tard (moniteur, lecteur de cassette ou de disquette), on arrivait à une somme dépassant largement le budget que l'on s'était imparti pour finalement disposer d'un matériel n'offrant pas toutes les possibilités auxquelles on s'attendait. Avec Amstrad, plus d'économies de bouts d'octets ! Le système que l'on achète est directement utilisable et parfaitement autonome. Fin 1985, le CPC 464, seul produit disponible jusque-là, a fait deux petits... ou plutôt deux grands : les modèles CPC 664 et CPC 6128. Les trois modèles de la gamme Amstrad sont tous construits autour du même microprocesseur, le bon vieux Z80 qui a fait depuis longtemps ses preuves et qui reste parmi les plus performants des microprocesseurs 8 bits.

TROIS MODÈLES, UNE MÊME PHILOSOPHIE MAIS QUELQUES DIFFÉRENCES...

Au-delà de leur *design* (les couleurs du clavier s'assagissent sensiblement sur les derniers modèles), il est incontestable que la différence essentielle concerne le support de mémoire morte intégré à chacun de ces ordinateurs : cassettophone sur le CPC 464, lecteur de disquette (format 3 pouces d'Hitachi) sur les modèles CPC 664 et CPC 6128. En plus de sa plus grande fiabilité, le lecteur de disquette permet un accès aux fichiers incomparablement plus rapide que le magnétocassette. Il convient toutefois de noter une limitation étonnante pour un produit de cette qualité : le BASIC Amstrad ne comporte aucune instruction ou fonction destinée à la constitution et à la gestion de fichiers à accès aléatoire ! On devra donc se contenter des fichiers séquentiels... Amstrad commercialise d'autre part l'unité de disquette DDI – 1 (munie d'une interface) qui fait (presque) ressembler le CPC 464 à ses grands frères. Réciproquement, il est possible de connecter un magnétophone standard (prise DIN) sur les modèles CPC 664 et CPC 6128.

Les trois modèles disposent de 64K de mémoire vive dont environ 43K sont disponibles pour les programmes utilisateur, 16K étant réser-

vés à la mémoire écran. Le CPC 6128 dispose de 64K de mémoire RAM supplémentaires utilisables comme disque virtuel. L'interpréteur BASIC se trouve, sur les trois machines, dans les 32K de mémoire morte (ROM). Cette mémoire morte contient également, dans le cas des modèles CPC 664 et CPC 6128, le système d'exploitation du disque AMSDOS. Le rôle principal d'un système d'exploitation (de disque) est d'établir une série de conventions d'entrées/sorties entre l'unité centrale et le contrôleur du disque. AMSDOS est assez rudimentaire. Cependant, Amstrad livre également un second système d'exploitation (sur disquette) pouvant se substituer à AMSDOS. Il s'agit du fameux CP/M de Digital Research. Le modèle CPC 6128 est vendu avec un troisième système d'exploitation (en plus des deux premiers !), CP/M+, qui, comme son nom l'indique, procure un *plus* (indispensable) à l'utilisateur.

LES SYSTÈMES D'EXPLOITATION DU DISQUE : AMSDOS, CP/M 2.2 et CP/M+

Les trois systèmes d'exploitation ne sont pas concurrents mais bien complémentaires. Selon l'application que l'on souhaite développer, on aura grand intérêt à choisir le système le mieux adapté à cette application.

AMSDOS est assez rudimentaire en ce sens qu'il ne possède aucune commande permettant d'effectuer une gestion de fichiers digne de ce nom (copie de fichier, copie de disque, etc.). Il ne permet pas davantage le formatage des disquettes, opération préalable à toute utilisation de celles-ci. Il dispose par contre d'une particularité très intéressante : les commandes AMSDOS sont directement accessibles à partir du BASIC. Ce système d'exploitation permet donc une utilisation optimale des possibilités graphiques et sonores du BASIC Amstrad. Une rubrique de ce livre (Chapitre 3) est consacrée aux commandes AMSDOS.

Les programmes BASIC sont effacés de la mémoire centrale dès que l'on se trouve sous CP/M. CP/M et BASIC sont totalement indépendants. Il existe une commande du système AMSDOS (donc accessible en BASIC) permettant de passer sous CP/M. Réciproquement, la commande CP/M ''AMSDOS'' provoque le retour sous le système d'exploitation résident. CP/M ne permettant pas l'accès direct aux commandes BASIC, il est impossible d'exploiter, sous ce système, les possibilités graphiques et sonores de l'Amstrad. En revanche, CP/M

dispose de toutes les commandes de copie et de formatage nécessaires. Il permet également de gérer l'interface série et les entrées/sorties concernant l'ensemble des unités périphériques. CP/M dispose enfin de quelques utilitaires comme un éditeur de texte (ED), ou un mini-assembleur (ASM). La véritable programmation en langage d'assemblage nécessite cependant de recourir à un logiciel plus performant, comme par exemple le ZEN d'Avalon Software (distribué en France par Sybex).

L'un des avantages de pouvoir exploiter Amstrad sous CP/M est que l'on a ainsi accès à une foule de logiciels (ou de langages) professionnels performants qui ont été développés sous ce système d'exploitation (lorsque ces logiciels ont été transférés dans le format encore peu répandu des disquettes Amstrad). Signalons par exemple que la société Fraciel a adapté sur Amstrad une version du Turbo Pascal. Cependant, un nombre important de ces logiciels professionnels se trouvent un peu à l'étroit dans les 64K de mémoire RAM des modèles CPC 464 et CPC 664. Ce n'est plus nécessairement vrai dans les 128K du CPC 6128. CP/M+ permet de gérer ces 128K et étoffe considérablement les possibilités de CP/M, en ce qui concerne tant les commandes nouvelles que la zone laissée libre à l'utilisateur.

LE BASIC LOCOMOTIVE, UN BASIC QUI A PRIS LE BON WAGON...

C'est la société Locomotive Software qui a développé le BASIC résident de l'Amstrad. Ce BASIC supporte tout à fait la comparaison avec la référence qu'est devenu le BASIC Microsoft, au moins en ce qui concerne ses versions les plus populaires (MSX, Thomson, ...). Si l'on peut regretter l'absence de *sprites* (qui facilitent l'animation) et celle d'instructions permettant le tracé direct de cercles (CIRCLE) ou de rectangles (BOX ou LINE), toutes les autres instructions graphiques essentielles sont présentes (au moins sur les modèles CPC 664 et 6128), avec en plus la possibilité de gérer l'écran en mode multifenêtre (pas tout à fait à la façon du Macintosh tout de même) grâce aux instructions WINDOW et WINDOW SWAP. La palette de couleurs dispose de 27 couleurs différentes et la résolution graphique est très bonne (640×200 pixels, en haute résolution) pour une machine de cette gamme. Le générateur de son est un synthétiseur à trois canaux disposant de huit octaves (General Instrument, AY3 8910). Ce circuit est également utilisé par un grand nombre d'ordi-

nateurs concurrents. Cependant, l'interpréteur BASIC de l'Amstrad a été conçu pour permettre une exploitation maximale des possibilités de ce circuit. La contrepartie de ce contrôle très précis est une certaine complexité au plan de la programmation des sons. Une rubrique très détaillée de cet ouvrage regroupe les instructions et fonctions afférentes à la gestion du son (Chapitre 3, le son Amstrad). Signalons enfin une particularité très intéressante du BASIC Locomotive : l'ordinateur possède une horloge interne à laquelle sont associés quatre chronomètres indépendants. Des instructions d'interruptions logicielles permettent de simuler une utilisation multitâche de l'ordinateur.

Les modèles CPC 664 et CPC 6128 disposent de quelques instructions BASIC supplémentaires par rapport au modèle antérieur. Ces instructions nouvelles sont essentiellement dirigées vers le graphisme. D'autres instructions déjà existantes ont bénéficié, sur les modèles récents, de quelques options supplémentaires (PEN, DRAW, PLOT, ...).

ORGANISATION DE L'OUVRAGE

Ce livre s'adresse tant aux débutants qu'aux programmeurs confirmés. Ceux-ci y trouveront la syntaxe complète de toutes les commandes, fonctions et instructions du BASIC Amstrad (et de AMSDOS). De nombreux programmes de démonstration, simples ou élaborés, illustrent un grand nombre d'instructions.

Les débutants trouveront, au Chapitre 2, le rappel des notions élémentaires concernant la programmation en BASIC. Des informations complémentaires sont données tout au long de l'ouvrage. La notion de fichier est par exemple développée sous les rubriques "Commandes AMSDOS" et "OPENIN" ; les notations binaire et hexadécimale sont illustrées et commentées en regard des fonctions BIN$ et PEEK.

Toute classification étant arbitraire et souffrant nécessairement d'exceptions, l'organisation suivante a été adoptée dans cet ouvrage : les instructions, fonctions et commandes du BASIC Amstrad sont présentées par ordre alphabétique, certains regroupements ayant cependant été effectués, soit pour des raisons thématiques, soit pour des raisons de syntaxe ou d'utilisation. Par exemple, les instructions LEFT$ et RIGHT$, qui effectuent le même type d'opération et qui s'emploient dans des syntaxes identiques, forment ici une seule et même rubrique.

De même, les instructions et fonctions classées sous la rubrique ''Gestion des erreurs'' sont toujours utilisées ensemble (et dans le même but) dans un programme.

Pour trouver une information concernant un sujet précis, la méthode la plus simple consiste à se reporter à l'index alphabétique donné à la fin de l'ouvrage. Des renvois fréquents à d'autres rubriques sont inclus dans la description de bon nombre d'instructions. Ce mode d'organisation ne nécessite pas, bien au contraire, une lecture linéaire de l'ouvrage. Selon ses connaissances et ses centres d'intérêt, le lecteur pourra l'aborder par une rubrique ou par une autre.

L'écriture du programme le plus simple met en jeu plusieurs instructions. Pour autant, il n'est pas nécessaire de connaître toutes les instructions du BASIC Amstrad pour pouvoir commencer à écrire de petits programmes. Les programmes d'application donnés dans ce livre ont été conçus dans cet esprit. Le programmeur chevronné pourra par exemple trouver décevant qu'aucun graphisme n'accompagne le programme de jeu illustrant la rubrique ''Opérateurs logiques''. Il n'aura par contre aucune difficulté à ajouter lui-même les instructions correspondantes. Le débutant aura quant à lui la satisfaction de comprendre, dans ses moindres détails, l'élaboration d'un programme complet n'utilisant que les instructions qu'il connaît (s'il a suivi l'ordre suggéré ci-après).

Les différentes rubriques alphabétiques sont regroupées ci-après par thèmes. En abordant successivement (et dans l'ordre) les rubriques correspondant à chacun de ces thèmes, le débutant pourra trouver une aide précieuse pour un apprentissage progressif de la programmation. Lorsqu'une rubrique contient plusieurs mots clés ou instructions, tous ne sont pas ici nécessairement répertoriés. Les parenthèses signalent des instructions devant normalement figurer dans le thème, mais dont il est préférable de réserver l'étude à un autre sujet.

COMMANDES ET INSTRUCTIONS D'USAGE GÉNÉRAL

AUTO, CAT, DELETE, EDIT, LIST, TRON et TROFF, NEW, RENUM, RUN, END, CONT, REM, STOP, CLEAR

STOCKAGE ET CHARGEMENT D'UN PROGRAMME SUR CASSETTE OU SUR DISQUE

CHAIN, LOAD, MERGE, SAVE, (Commandes AMSDOS)

MISE EN FORME DE L'AFFICHAGE A L'ÉCRAN OU SUR L'IMPRIMANTE

BORDER, CLS, COPYCHR$, CURSOR, DEC$, INK, LOCATE, MODE,
POS et VPOS, PRINT, PRINT USING
SPEED INK, SYMBOL, SYMBOL AFTER, SPC, TAB, WIDTH,
WINDOW, WINDOW SWAP
ZONE

OPÉRATIONS LOGIQUES, ARITHMÉTIQUES ET BASES DE CALCUL

MOD, DEF FN, fonctions arithmétiques, DEG et RAD, PI, RANDOMIZE
Opérateurs logiques
BIN$ et HEX$

INSTRUCTIONS DIRIGÉES VERS LE CLAVIER

INKEY, INKEY$, INPUT, LINE INPUT
KEY, KEY DEF, SPEED KEY

BRANCHEMENTS ET BOUCLES

IF...THEN...ELSE, GOTO, GOSUB/RETURN, ON...GOSUB et
ON...GOTO
FOR/NEXT, WHILE/WEND
Gestion des erreurs et DERR

ON BREAK CONT, ON BREAK GOSUB, ON BREAK STOP
(ON SQ GOSUB), (AFTER...GOSUB), (EVERY...GOSUB)

GESTION DU TEMPS

AFTER...GOSUB, DI, EI, EVERY...GOSUB, REMAIN, TIME

DÉCLARATION, ORGANISATION ET MANIPULATION DES DONNÉES

DEFtype, DIM, ERASE, FRE, LET
READ/DATA, RESTORE, (WRITE)

MANIPULATION DES CHAÎNES DE CARACTÈRES

ASC, INSTR, LEN, VAL
CHR$, LEFT$ et RIGHT$, LOWER$ et UPPER$, MID$, SPACE$, STRING$, STR$

FICHIERS

OPENIN, OPENOUT, CLOSEIN, CLOSEOUT, EOF, WRITE
Commandes AMSDOS

ADRESSAGE DIRECT DE LA MÉMOIRE EN BASIC

HIMEM, MEMORY, PEEK, POKE, @

GRAPHISME, SON ET MANETTES DE JEU

CLG, DRAW et DRAWR, FILL, FRAME, GRAPHICS PAPER, GRAPHICS PEN, MASK, MOVE et MOVER, ORIGIN, PLOT et PLOTR, TAG et TAG OFF, TEST et TESTR, XPOS et YPOS
Le son Amstrad
JOY

PROGRAMMATION EN LANGAGE MACHINE ET GESTION DES ENTRÉES/SORTIES

CALL, WAIT, INP et OUT

CONVENTIONS D'ÉCRITURE

Le format donné pour chaque instruction, commande ou fonction utilise les conventions d'écriture suivantes :

- le (ou les) mot clé (qui doit être tapé tel quel) est écrit en majuscules ;
- les éléments ou paramètres inclus entre les symboles < et > sont facultatifs (les signes < et > ne doivent naturellement pas être tapés).

Les nombres (en général des adresses) suivis de la lettre H symbolisent des nombres hexadécimaux. Dans un programme, la lettre H doit être remplacée par la fonction &H (voir la rubrique BIN$).

Lorsque aucune indication contraire n'est donnée, la rubrique (ou plus généralement l'information fournie) concerne aussi bien le CPC 464 que le CPC 664 ou le CPC 6128.

2

PROGRAMMER EN BASIC SUR AMSTRAD

La première partie de ce chapitre est destinée à initier le débutant à la conception et à l'élaboration d'un programme BASIC. Des informations complémentaires importantes sont par ailleurs données tout au long de l'ouvrage. Ce mode d'organisation permettra au lecteur qui suivrait l'ordre suggéré dans l'Introduction de mettre tout de suite en pratique les premières connaissances acquises.

Ce chapitre se poursuit par une description sommaire de l'éditeur Amstrad, outil fondamental pour l'écriture et la correction de lignes de programme. Un glossaire rappelle finalement la définition d'une vingtaine de mots courants du vocabulaire informatique.

NOTIONS ÉLÉMENTAIRES DE PROGRAMMATION EN BASIC

Supposons que l'on veuille calculer le résultat de l'opération suivante :

$$28{,}492 \times 4 \; / \; 5{,}39$$

L'ordinateur permet d'effectuer ce calcul de deux manières différentes. La première consiste à taper au clavier les caractères suivants (en respectant exactement la syntaxe) :

```
PRINT 28.492 * 4 / 5.39
```

Après avoir pressé la touche Enter (équivalente à la touche retour chariot du clavier de machine à écrire), on voit le résultat de l'opération s'afficher à l'écran :

```
21.1443414
```

Le même résultat peut être obtenu en procédant comme indiqué ci-dessous, c'est-à-dire en créant un (tout petit) programme :

```
10 PRINT 28.492 * 4 / 5.39
20 END
```

Après avoir tapé ces deux lignes (chacune devant là aussi se terminer par un retour chariot), écrivons le mot :

```
RUN
```

et pressons une fois encore la touche Enter. Si tout s'est bien passé, le résultat de l'opération doit de nouveau s'afficher à l'écran.

La première manière de procéder consiste à utiliser l'ordinateur en mode direct (encore appelé mode immédiat ou mode commande). La seconde utilisation correspond au mode programme. On remarque que, dans l'un et l'autre modes, le signe multiplié et les virgules séparant la partie entière de la partie décimale des nombres doivent être respectivement remplacés, en langage BASIC, par un astérisque (*) et par un point (.). D'autre part, en mode direct, il ne suffit pas de taper :

```
28.492 * 4 / 5.39 =
```

pour que le résultat de l'opération soit affiché. L'ordinateur sait calculer l'expression demandée mais il ne sait pas quoi faire du résultat. Il prend ici le signe égal comme un symbole d'affectation d'une valeur (l'expression à calculer) à une variable ou à une constante qui ne lui est pas fournie. Il est par contre possible d'écrire :

```
C = 28.492 * 4 / 5.39
PRINT C
```

Dans un premier temps, le résultat du calcul est assigné comme valeur à la variable C. Il est ensuite demandé à l'ordinateur d'afficher à l'écran (c'est l'objet du mot PRINT) la valeur de cette variable.

En algèbre, les deux relations :

$$C = 28.492 * 4 / 5.39$$

et

$$28.492 * 4 / 5.39 = C$$

sont correctes et équivalentes. Il n'en est pas nécessairement de même en BASIC. Supposons que la deuxième ligne soit entrée et que l'on tape ensuite :

```
PRINT C
```

La valeur 0 sera alors affichée. Pourquoi ?

Revenons au petit programme de démonstration. On remarque

qu'en mode programme, chaque ligne commence par un numéro. C'est une des caractéristiques obligatoires de ce mode. Lorsque nous écrivons :

```
28.492 * 4 / 5.39 = C
PRINT C
```

l'ordinateur interprète la première ligne comme étant une ligne de programme et son analyseur syntaxique la transcrit en :

```
28  0.492 * 4 / 5.39 = C
```

c'est-à-dire qu'à la ligne 28 d'un programme en train d'être écrit, le résultat de l'opération (0,492 × 4 / 5,39), c'est-à-dire la valeur 0,3651, est assigné à la variable C. Lorsque l'on demande ensuite à l'ordinateur d'afficher à l'écran la valeur de C (grâce à PRINT C), on revient alors en mode direct puisque PRINT C n'est pas précédé d'un numéro de ligne. On pourrait penser que la valeur 0,3651 (et non 0) devrait dans ce cas s'afficher. Il n'en est pas ainsi pour la raison suivante : il y a bien en mémoire centrale de l'ordinateur un programme (constitué d'une seule ligne ayant le numéro 28), mais pour que ce programme réalise les opérations auxquelles il est destiné, sa présence en mémoire est une condition nécessaire mais non suffisante. Il faut en outre demander à l'ordinateur de l'exécuter, ce que l'on fait en écrivant en mode direct, une fois que toutes les instructions ont été entrées, le mot RUN (comme dans le premier exemple). Tant que le programme n'a pas été exécuté, toutes les variables sont égales à 0, d'où la valeur de C.

PRINT, END et RUN sont trois des quelque 140 mots clés (ou mots réservés) constituant le langage de programmation BASIC utilisé sur l'Amstrad. Ce vocabulaire, d'origine anglo-saxonne, est généralement assez signifiant par rapport à la fonction réalisée, pour peu que l'on possède quelques notions élémentaires d'anglais. Dans le cas contraire, le handicap est de peu d'importance ; il est très rapidement surmonté par un minimum de pratique de la programmation.

Le mot PRINT commande à l'ordinateur d'effectuer une action ; il s'agit donc d'une *instruction*. PRINT peut être utilisé aussi bien en mode programme qu'en mode direct. On parlera d'une instruction programmable et exécutable. D'autres instructions sont spécifiquement réservées au mode programme (instructions programmables).

Au contraire, certains mots clés tels que RUN ne peuvent être employés qu'en mode direct. On réservera à ces derniers le nom de *commandes*. Une instruction peut être formée de plusieurs mots clés, comme par exemple ON ERROR GOTO.

Une autre catégorie de mots clés doit être mentionnée : il s'agit des *fonctions*, qui sont généralement exécutables et programmables et qui peuvent être soit algébriques (COS, EXP, CSNG, ...), soit propres à la programmation (PEEK, STR$, etc.). Une fonction se caractérise par le fait qu'elle renvoie une valeur représentant le résultat d'une opération préprogrammée dans les circuits de l'ordinateur.

Il faut enfin savoir qu'en mode direct une seule instruction ou commande peut être exécutée à la fois, ce qui signifie qu'il faut attendre le résultat de la première (qui est affiché dès que la touche Enter est pressée) avant d'entrer la seconde.

Les règles énumérées ci-dessous doivent être respectées lors de l'écriture d'un programme en BASIC Amstrad. Ces règles sont d'ailleurs valables pour la grande majorité des autres versions de ce langage de programmation :

- Une ligne de programme peut comporter plusieurs instructions ; celles-ci doivent alors être séparées par deux points (:).

- Une ligne de programme ne peut contenir plus de 255 caractères, y compris les espaces et le caractère retour chariot terminal qui n'est pas imprimé mais qui sert à entrer la ligne écrite en mémoire centrale de l'ordinateur.

- Chaque ligne de programme doit être numérotée (entre 0 et 65529). Les numéros indiquent l'ordre dans lequel les lignes sont enregistrées en mémoire.

- Les numéros de lignes doivent respecter la logique d'exécution du programme. Par exemple, l'instruction destinée à faire imprimer le résultat d'un calcul ne doit pas être rencontrée par le programme avant l'instruction qui réalise ce calcul.

- L'exécution du programme commence à la ligne ayant le numéro le plus faible et continue par ordre de numéros croissants (sauf lorsqu'un branchement est rencontré).

- L'incrément entre deux lignes successives est quelconque. Il n'a pas non plus besoin d'être constant entre toutes les lignes.

- L'ordre d'entrée des lignes en mémoire centrale de l'ordinateur peut être quelconque lors de l'écriture du programme (tant que ces lignes sont correctement numérotées). Il est indifférent d'écrire d'abord la ligne 100 et ensuite la ligne 10, ou l'inverse.
- L'écriture d'une ligne possédant le même numéro qu'une ligne déjà existante dans le programme efface l'ancienne. En conséquence, pour supprimer une ligne de programme, il suffit d'entrer seulement son numéro de ligne.
- Un programme qui se trouve en mémoire centrale de l'ordinateur s'efface lorsque l'alimentation de la machine est coupée. Il est cependant possible de conserver le programme (ou les données) sur un support mémoire permanent (bande magnétique, disquette) d'une unité périphérique (magnétophone, unité de disque) connectée à l'ordinateur (ou directement intégrée à celui-ci, dans le cas de l'Amstrad).
- Lorsqu'un programme se trouvant sur un support mémoire permanent est appelé, une copie de ce programme est *chargée* en mémoire centrale de l'ordinateur. La disquette ou la bande magnétique n'est pas effacée pour autant. Si l'on modifie ce programme, on modifie donc la copie. Pour que ces modifications soient également conservées sur la version stockée sur le support permanent, il est nécessaire de *sauvegarder* ce programme modifié, c'est-à-dire de réaliser l'opération inverse (transfert du programme de la mémoire centrale de l'ordinateur vers le support magnétique).

ÉCRITURE ET MODIFICATION D'UN PROGRAMME : L'ÉDITEUR AMSTRAD

Les ordinateurs Amstrad ne disposent que d'un éditeur de ligne ne permettant pas l'accès en tout point de l'écran. Cela signifie qu'il est toujours nécessaire de préciser le numéro de la ligne que l'on veut modifier et qu'il ne suffit pas, comme pour un éditeur pleine page, de positionner le curseur sur la ligne de programme en question. Cet éditeur met en jeu les touches suivantes :

- Retour chariot (Enter).

- Déplacement du curseur (quatre touches marquées d'une flèche, au-dessus (CPC 464 et CPC 664) ou au-dessous (CPC 6128) du pavé numérique).
- SHIFT et CTRL.
- CLR (effacer) et DEL (délétion) en haut et à droite du clavier machine à écrire.
- COPY, touche située au centre du pavé consacré aux touches de déplacement du curseur (CPC 464 et CPC 664) ou à gauche de la barre d'espacement (CPC 6128).

L'utilisation de l'éditeur va être illustrée en décrivant la procédure à suivre pour modifier une ligne de programme existante (celle-ci est supposée se trouver déjà en mémoire centrale, c'est-à-dire qu'elle a été validée par un retour chariot). Supposons donc que la ligne suivante :

```
50 PRINT "Illustrtaion de l'éditeur AMSTRAD"
```

ait été écrite et que l'on veuille remplacer le mot "Illustrtaion" par "Illustration". Pour ce faire, le plus simple consiste à utiliser la touche COPY, après avoir dédoublé le curseur (obtention d'un curseur de copie). Le dédoublement du curseur est obtenu en appuyant simultanément sur la touche SHIFT et sur l'une quelconque des touches de déplacement du curseur. En maintenant la touche SHIFT appuyée, le curseur de copie est ensuite déplacé sur l'écran jusqu'à atteindre le premier caractère de la ligne à modifier, soit le chiffre 5 dans l'exemple choisi. Ce caractère s'affiche alors en vidéo inverse.

1. Taper la touche COPY pour recopier les premiers caractères de la ligne ne devant pas être modifiés. Les caractères recopiés s'affichent à la position du curseur principal, à mesure que le curseur de copie se déplace. Dans notre exemple, nous appuierons sur la touche COPY jusqu'à ce que le curseur de copie se place sur la deuxième lettre T du mot "Illustrtaion". Le début de la ligne '50 PRINT "Illustr' aura alors été recopié au niveau du curseur principal.

2. Relâcher la touche COPY et entrer les caractères de remplacement. Ceux-ci s'inscriront au niveau de la position du curseur prin-

cipal (ligne en cours de recopie) sans que le curseur de copie soit déplacé. Dans notre exemple, nous taperons successivement les lettres A et T.

3. Appuyer deux fois sur la touche flèche droite (déplacement du curseur) en maintenant la touche SHIFT enfoncée. Le curseur de copie saute alors les lettres erronées T et A pour se placer sur la lettre I du mot ''Illustrtaion''. Ce faisant, rien de nouveau ne s'écrit sur la ligne en cours de recopie (curseur principal). Les touches flèche combinées avec la touche SHIFT permettent en effet de positionner le curseur de copie (sans provoquer de recopie lors du déplacement).

4. Appuyer sur la touche COPY pour recopier la fin de la ligne. Une fois celle-ci atteinte, taper ENTER pour valider la modification. Ainsi, la ligne recopiée remplace l'ancienne en mémoire centrale de l'ordinateur.

Il est également possible de procéder de la manière suivante : après avoir dédoublé le curseur, la ligne entière est recopiée au moyen de la touche COPY, sans que la copie soit validée, dans un premier temps, par la touche ENTER. Les modifications sont alors apportées sur la ligne recopiée mais non validée (c'est-à-dire au niveau du curseur principal). Pour ce faire, on peut utiliser toutes les touches d'édition normale, DEL, CLR, déplacement du curseur (combinée éventuellement avec la touche CTRL pour se placer en début ou en fin de ligne). Une fois les modifications effectuées, la ligne est validée en tapant la touche ENTER. Ce mode d'édition est automatiquement sélectionné par l'ordinateur lorsque certaines erreurs sont rencontrées au cours de l'exécution d'un programme. La ligne contenant l'erreur est en effet automatiquement recopiée et le curseur (non dédoublé) est placé au niveau du premier caractère de cette ligne. Les corrections peuvent alors être apportées comme si la ligne n'avait pas encore été validée.

Adresse

Emplacement d'une zone mémoire quelconque. Chaque octet de la mémoire possède ainsi une adresse permettant de le localiser.

Argument

Nom donné à la valeur d'un paramètre transmis par un programme appelé (ou par une fonction) à un programme appelant.

ASCII

Code standard permettant de représenter les caractères par des nombres.
Un fichier ASCII est un fichier texte dans lequel les caractères sont représentés par leur code ASCII.

Bit

Chiffre binaire 0 ou 1 (contraction de *binary digit*).

Concaténation

Assemblage de deux chaînes de caractères en une seule. La concaténation de deux chaînes s'effectue au moyen de l'opérateur +.

Chaîne

Séquence de caractères (255 au maximum) que l'ordinateur traite comme entité unique ou multiple selon l'instruction utilisée.

Défaut (valeur par)

Valeur retenue par l'ordinateur pour un paramètre, lorsque celle-ci n'est pas précisée par ailleurs.

Fichier

Ensemble d'informations stockées sous un nom donné.

Instruction

Mot (ou ensemble de mots) clé commandant à l'ordinateur de réaliser une opération élémentaire.

Langage machine

Ensemble d'instructions directement exécutables par le microprocesseur, par opposition aux instructions écrites en BASIC, qui doivent être d'abord *interprétées*.

Mémoire RAM

Partie de la mémoire de l'ordinateur permettant un accès en mode lecture et écriture. Les informations se trouvant en mémoire RAM sont effacées dès que l'ordinateur est mis hors tension.

Mémoire ROM

Mémoire dont le contenu ne peut qu'être lu (et non modifié) par le programmeur. Les informations stockées en mémoire ROM sont permanentes.

Mot clé

Mot prédéfini dans un langage de programmation (vocabulaire).

Octet

Groupe de huit bits. Chaque bit pouvant valoir 0 ou 1, un octet peut avoir 256 valeurs différentes.

Opérateur

Symbole ou nom permettant d'effectuer une opération. Il s'agit d'un signe (+, −, *, etc.) pour une opération arithmétique, ou d'un mot (AND, OR, XOR, etc.) pour une opération logique.

Programme

Ensemble d'instructions permettant à l'ordinateur de réaliser une tâche définie par le concepteur (du programme).

Système d'exploitation

Logiciel (programme) qui contrôle l'exécution des autres programmes.

Variable

Paramètre ayant une valeur modifiable. Chaque variable doit avoir un nom. Le nom d'une variable de chaîne doit toujours se terminer par le symbole $. Un nom de variable doit toujours commencer par une lettre.

3

INSTRUCTIONS, COMMANDES ET FONCTIONS BASIC DE L'AMSTRAD CPC 464, CPC 664 ET CPC 6128 COMMANDES AMSDOS

AFTER...GOSUB

Instruction

L'instruction AFTER...GOSUB provoque un branchement vers un sous-programme après qu'un délai, dont la durée est un paramètre de l'instruction, se soit écoulé.

FORMAT

AFTER n1 <,n2> GOSUB numéro de ligne

- **n1** est un nombre entier spécifiant le délai au bout duquel le branchement doit s'effectuer. L'unité de temps est définie comme étant égale à 0,02 seconde. La valeur 100 correspondra donc à une attente de 2 secondes.
- **n2** est un nombre entier (0 à 3) spécifiant le numéro du chronomètre auquel s'applique l'instruction. L'Amstrad dispose de quatre chronomètres indépendants. La valeur par défaut du paramètre n2 est zéro.
- Le **numéro de ligne** est celui de la première ligne du sous-programme vers lequel sera effectué le branchement une fois le délai écoulé. Ce sous-programme doit se terminer par une instruction RETURN.

COMMENTAIRES

Un sous-programme peut être associé à chacun des quatre chronomètres indépendants. L'exécution d'un sous-programme démarre lorsque les trois conditions suivantes sont simultanément réalisées :

- Le délai spécifié dans l'instruction AFTER...GOSUB est écoulé.
- Il n'y a pas de branchement en attente spécifié par un chronomètre ayant un ordre de priorité plus élevé. Cet ordre de priorité va du chronomètre n° 3 (priorité la plus élevée) vers le chronomètre n° 0.

- Il n'y a pas d'instruction en cours d'exécution. Dans le cas contraire, il faudra attendre la fin de son exécution avant que le sous-programme ne soit effectivement lancé. Il faut avoir cette condition présente à l'esprit lorsqu'une instruction AFTER...GOSUB est utilisée dans un programme faisant appel aux instructions d'entrée clavier telles que INPUT ou LINE INPUT.

Un sous-programme peut être associé à chacun des quatre chronomètres disponibles. Au retour du sous-programme, l'exécution est redirigée vers l'instruction du programme principal qui suit celle où s'est produite l'interruption (et non vers l'instruction se trouvant après AFTER...GOSUB).

Les chronomètres sont également utilisés par les instructions EVERY...GOSUB. L'exécution de l'une ou l'autre de ces instructions réinitialise le décompte du temps dépendant du chronomètre spécifié.

La fonction REMAIN annule le branchement déclaré par une instruction AFTER...GOSUB (ou EVERY...GOSUB). Les instructions DI et EI permettent respectivement d'interrompre et de reprendre le décompte du temps.

EXEMPLE DE PROGRAMME

Le programme ci-dessous propose un petit exercice de calcul mental. Il s'agit de trouver, en un temps limité, le résultat de la multiplication de deux nombres entiers (compris entre 11 et 26) générés au hasard (lignes 40 à 60). Le produit de ces deux nombres est stocké dans la variable R (ligne 70). Deux chronomètres sont utilisés, l'un (n° 0) étant géré par une instruction EVERY...GOSUB (ligne 110), l'autre (chronomètre 1) dépendant d'une instruction AFTER...GOSUB (ligne 120).

```
10 CLS
20 T=10
30 W=0
40 RANDOMIZE TIME
50 R1=INT(15*RND(1))+11
60 R2=INT(15*RND(1))+11
70 R=R1*R2
80 PRINT "Tapez une touche,puis entrez l
e resultat"
90 LOCATE 15,6
100 PRINT r1;"x";r2;"=?"
110 EVERY 50,0 GOSUB 240
```

→

```
120 AFTER 550,1 GOSUB 340
130 IF W=1 THEN 200
140 V$=INKEY$ : IF V$="" GOTO 130
150 LOCATE 25,7
160 INPUT V
170 IF W=1 THEN PRINT "petit malin ! il
ne faut pas bloquer le chronometre penda
nt que l'on reflechit" : GOTO 200
180 IF V=R THEN PRINT "BRAVO" ELSE PRINT
 "FAUX :";R
190 D1=REMAIN(1) : D2=REMAIN(0)
200 LOCATE 1,25
210 PRINT "Tapez une touche pour continu
er";
220 A$=INKEY$ : IF A$="" THEN 220 ELSE 1
0
230 '*********************************
240 '    SOUS-PROGRAMME CHRONOMETRE
250 '*********************************
260 LOCATE 20,15
270 INK 1,6
280 PRINT "Chronometre :";T
290 INK 1,25
300 T=T-1
310 RETURN
320 '
330 '*********************************
340 '       SOUS-PROGRAMME ECHEC
350 '*********************************
360 C=REMAIN(0) : W=1
370 LOCATE 1,10
380 PRINT "Pas assez rapide"
390 PRINT
400 PRINT R1;"x";R2;"=";R
410 RETURN
```

Le branchement vers le sous-programme débutant à la ligne 240 est effectué chaque seconde (voir l'instruction EVERY...GOSUB). Ce sous-programme est destiné à afficher le défilement du chronomètre. Un effet de flash est associé à ce défilement (lignes 270 et 290). Au bout de $0{,}02 \times 550 = 11$ secondes, l'exécution est redirigée par l'instruction AFTER...GOSUB de la ligne 120 vers le sous-programme commençant à la ligne 340, dont l'objet est d'afficher le résultat de l'opération. Ce branchement ne s'effectue que si le résultat proposé par le joueur n'a pas été saisi avant la fin du temps imparti. Dans le cas contraire, la fonction REMAIN(1) de la ligne 190 permet d'annu-

ler le branchement. La manière dont est saisi le résultat entré au clavier (lignes 110 à 140) mérite quelque attention.

Après que l'instruction AFTER...GOSUB a été exécutée, le programme boucle sur les lignes 130 et 140, dans l'attente d'une entrée clavier. Une fois celle-ci effectuée (en tapant n'importe quelle touche), le résultat proposé est saisi au moyen d'une instruction INPUT. La boucle se révèle ici indispensable ; si les lignes 130 et 140 étaient supprimées, il serait impossible d'afficher le temps restant, dans l'attente de l'entrée du résultat au clavier. Les chronomètres continueraient bien de tourner, mais les branchements ne pourraient s'effectuer avant la fin de la saisie.

La variable W initialisée à la ligne 30 et utilisée aux lignes 130, 170 et 360 permet de rediriger l'exécution en fonction des sous-programmes réellement exécutés. Les chronomètres continuant de tourner pendant la saisie (même si le défilement est interrompu à l'écran), il est possible de savoir si le résultat a bien été entré dans le délai imparti (auquel cas le sous-programme commençant à la ligne 340 n'aura pas été exécuté ; la variable W vaudra donc 0 et le message de la ligne 170 ne sera pas affiché).

La fonction ASC donne la valeur numérique du code ASCII du premier caractère d'une chaîne spécifiée comme argument.

FORMAT

ASC(X$)

- **X$** est une expression alphanumérique (chaîne de caractères).

Exemples

ASC(''code'')

ASC(X$)

en supposant qu'une valeur ait été préalablement fournie à la variable X$.

RÈGLES DE SYNTAXE ET COMMENTAIRES

Le jeu de caractères utilisé par l'Amstrad se compose de caractères alphabétiques, numériques, de signes de ponctuation, d'opérateurs arithmétiques et d'un certain nombre de caractères spéciaux (dièse, dollar, etc.). Chacun de ces caractères possède un numéro de code (qui n'est pas particulier à l'Amstrad) qui est un nombre compris entre 0 et 127 (code ASCII standard). D'autres caractères plus particuliers (les caractères graphiques par exemple), qui n'ont pas nécessairement de signification en BASIC, ont un numéro de code compris entre 128 et 255 (code ASCII étendu). La liste des codes ASCII est donnée en annexe du guide de l'utilisateur livré avec l'ordinateur.

La fonction ASC(X$) a précisément pour objet de retourner le numéro de code ASCII du premier caractère de la chaîne donnée comme valeur à X$. Si cette chaîne est nulle, un message d'erreur ''Improper argument in xxx'' (Argument incorrect en xxx) est affiché,

xxx correspondant au numéro de la ligne de programme où l'erreur a été décelée.

La fonction CHR$ effectue l'opération inverse de celle qui est réalisée par la fonction ASC, en renvoyant le caractère correspondant à un numéro de code ASCII donné.



Commande

La commande AUTO génère automatiquement des numéros de lignes croissants et d'incrémentation constante lors de chaque activation de la touche Entrée (RETURN).

FORMAT

AUTO < numéro > < , < incrément > >

- **numéro** indique le numéro de la première ligne générée.
- **incrément** est la différence entre deux numéros de lignes consécutifs.

La valeur par défaut de ces paramètres est 10.

RÈGLES DE SYNTAXE ET COMMENTAIRES

La commande AUTO est habituellement utilisée lors de l'écriture d'un programme. Elle évite d'avoir à frapper les numéros précédant chaque ligne d'instruction. Selon l'option choisie, l'action de la commande AUTO est la suivante :

Numéro	Incrément	Exemple	Résultat
spécifié	spécifié	AUTO 10,20	Génère des numéros de lignes commençant à la ligne 10 et espacés de 20 en 20 (10,30,50,...).
omis	omis	AUTO	Génère des numéros de lignes commençant à la ligne 10 et espacés de 10 en 10 (10,20,30,...).
spécifié	omis	AUTO 100	Prend comme incrément par défaut la valeur 10 (100,110,120,...).
omis	spécifié (précédé d'une virgule)	AUTO ,100	La numérotation commence à 10, en utilisant pour incrément la valeur de l'argument spécifié (10,110,210,...).

Lorsque la commande AUTO génère un numéro de ligne identique à celui d'une ligne existant déjà par ailleurs dans le programme, le numéro généré est immédiatement suivi d'un astérisque (*). Sur le CPC 664, la ligne entière déjà existante est réécrite après le numéro. Si l'on ne veut pas effacer l'ancienne ligne, il suffit de taper de nouveau la touche Entrée. Le numéro suivant sera généré et l'ancienne ligne conservée. Si de nouvelles instructions sont écrites après l'astérisque et que la ligne est entrée, l'ancienne ligne sera perdue.

Pour sortir du mode AUTO et revenir au mode direct, il suffit d'activer la touche Esc. La ligne en cours d'écriture sera perdue.

BIN$ et HEX$ &X et &H

Les fonctions BIN$ et HEX$ convertissent un nombre décimal (base 10) respectivement en un nombre binaire (base 2) et en un nombre hexadécimal (base 16). Ces fonctions renvoient une valeur de chaîne. Les fonctions &X et &H effectuent les conversions inverses et retournent respectivement la valeur décimale d'un nombre binaire et hexadécimal.

FORMATS

BIN$(expression numérique à convertir < ,format >)

HEX$(expression numérique à convertir < ,format >)

- L'**expression numérique à convertir** doit avoir une valeur n comprise entre −32768 et 65535. Lorsqu'elle est négative, ces fonctions prennent comme argument (65536 − n). Les nombres à convertir sont automatiquement arrondis à l'entier le plus proche.
- Le paramètre optionnel **format** définit le format dans lequel sera affiché le résultat de la conversion. Autant de zéros non significatifs seront ajoutés à la gauche du nombre converti pour respecter ce format. Par contre, le paramètre format ne sera pas respecté si le nombre de chiffres qu'il stipule est inférieur à ceux qui forment la valeur convertie.

Exemples

```
PRINT HEX$(8000,6)
```

Le résultat affiché est 001F40.

```
PRINT HEX$(8000,2)
```

Le résultat affiché est 1F40 (le format n'est donc pas respecté).

&X nombre binaire

&H nombre hexadécimal

Exemples

```
PRINT &H7F
```

La valeur 127 est affichée à l'écran.

```
PRINT &X1000
```

La valeur 8 est affichée.

COMMENTAIRES

Les nombres exprimés en formats binaire ou hexadécimal constituent le vocabulaire de base de l'ordinateur. Si l'on veut s'adresser directement à celui-ci sans passer par l'intermédiaire d'un interprète (ce qui est une des fonctions du langage BASIC), il faut utiliser ce vocabulaire. La mémoire centrale de l'ordinateur est constituée d'un ensemble de circuits électroniques qui n'ont chacun que deux positions possibles (ouvert ou fermé). C'est la raison pour laquelle l'ordinateur utilise, au niveau fondamental, le système binaire (0 et 1) pour représenter un *bit*, contraction anglo-saxonne de *binary digit*, nombre binaire. Pour des raisons d'efficacité, le microprocesseur groupe les bits par huit, chaque groupe constituant un octet (*byte*). Chaque bit ayant deux positions possibles (0 ou 1), un octet peut avoir $2^8 = 256$ configurations différentes. Certaines données, telles que le code d'un caractère, peuvent être stockées sur un seul octet. D'autres nécessitent deux octets. Chaque octet est identifié par son adresse (de 0 à 65535) dans la mémoire de l'ordinateur. Un programme, des variables et des constantes sont stockés sous la forme d'une suite d'octets.

La notation hexadécimale utilise les chiffres 0 à 9 et les lettres A à F. La lettre F en hexadécimal correspond au nombre décimal 15. Le préfixe &X sert à convertir un nombre hexadécimal en sa valeur décimale, le préfixe &H étant utilisé pour les conversions de nombres exprimés en notation binaire. Dans ce livre, un nombre exprimé en notation hexadécimale est suivi, par convention, de la lettre H (sauf lorsqu'il est précédé, comme c'est le cas dans un programme, du préfixe &H).

Les cinq lignes de programme suivantes permettent d'afficher à l'écran la correspondance entre la notation décimale, hexadécimale et binaire de 20 nombres consécutifs.

```
10 INPUT "Première valeur décimale à convertir";A
20 PRINT "Déc"," Hexa","Binaire"
30 FOR I=A TO (A+19)
40 PRINT I,HEX$(I),BIN$(I)
50 NEXT I
```

L'instruction BORDER définit la couleur de la bordure de l'écran.

FORMAT

BORDER numéro de couleur < ,numéro de couleur >

- Le **numéro de couleur** doit être compris entre 0 et 26.
- Le second numéro est optionnel. Lorsqu'il est fourni, la couleur dans laquelle s'affiche la bordure alterne entre les deux couleurs spécifiées.

COMMENTAIRES

Lorsque le second numéro de couleur est spécifié, la vitesse à laquelle se produit le changement de couleur dépend de l'argument de l'instruction SPEED INK en cours.

EXEMPLE DE PROGRAMME

Ce programme alterne les couleurs dans lesquelles s'affiche la bordure, en modifiant la vitesse de rotation et en associant un son différent à chaque couleur.

```
 5 T=100
10 FOR I=0 TO 26
20 IF T> =1000 THEN T=100 ELSE T=T*2
30 FOR J=1 TO T : NEXT J
40 BORDER I : SOUND 1,286+I
50 NEXT I
60 GOTO 10
```

Le programme peut être interrompu à tout moment au moyen de la touche Esc.

L'instruction CALL appelle un sous-programme écrit en langage machine.

FORMAT

CALL adresse < (liste de paramètres) >

- L'**adresse** est celle du premier octet du sous-programme ou celle d'un point d'entrée (appel d'un sous-programme du système d'exploitation).
- Les **paramètres** éventuellement spécifiés dans la liste doivent être séparés par des virgules.

COMMENTAIRES

Cette instruction est essentiellement réservée aux utilisateurs développant des programmes écrits en langage machine (langage d'assemblage).

La mémoire ROM de l'Amstrad contient quatre tableaux de points d'entrée destinés à maintenir la compatibilité avec les versions futures de l'ordinateur. Ces tableaux sont formés d'une suite d'instructions JUMP qui établissent les branchements vers les sous-programmes du système d'exploitation. Le contenu des tableaux est copié en mémoire RAM lors de l'initialisation du système. Alors que l'adresse même des sous-programmes en mémoire ROM pourra changer dans des versions futures, celle des points d'entrée ne sera pas modifiée (d'où l'intérêt de les utiliser si l'on veut accéder à ces sous-programmes). Le bloc JUMP principal se situe entre les adresses BB00H et BD37H.

La commande CAT permet d'établir le répertoire d'une cassette ou celui d'une disquette.

FORMAT

CAT

COMMENTAIRES

Le répertoire est la liste des fichiers se trouvant sur un support mémoire donné. Dans le cas d'une cassette, le répertoire est établi de manière séquentielle. Lorsqu'un fichier est trouvé, les indications suivantes sont affichées à l'écran :

nom du fichier – numéro du bloc – type du fichier

Le type du fichier est un caractère identifiant la manière dont le fichier a été sauvegardé :

- $ Programme BASIC non protégé.
- % Programme BASIC protégé.
- * Fichier ASCII (fichier texte dans lequel les caractères sont représentés par leur numéro de code ASCII).
- & Fichier binaire.

Le répertoire d'une disquette est affiché par ordre alphabétique (sur deux colonnes). Le nom de chaque fichier est suivi de sa taille (en kilo-octets, nombre arrondi à la valeur supérieure). La liste des noms de fichiers est précédée du numéro de l'unité de disque (A:, B:, etc.) et d'une identification donnée par l'utilisateur. Le nombre de kilo-octets restant disponibles sur la disquette est affiché à la suite de la liste.

Les instructions CHAIN et CHAIN MERGE chargent un programme se trouvant sur disque ou sur cassette. Le nouveau programme remplacera celui se trouvant préalablement en mémoire centrale (CHAIN) ou sera fusionné avec celui-ci (CHAIN MERGE).

FORMATS

CHAIN nom de fichier < ,numéro de ligne >

CHAIN MERGE nom de fichier < ,numéro de ligne > < ,DELETE numéro de ligne – numéro de ligne >

- Le **nom de fichier** est optionnel dans le cas d'une utilisation sur cassette. S'il n'est pas précisé, le premier fichier trouvé sera sélectionné.
- Le premier **numéro de ligne** (le seul dans le cas de CHAIN) correspond à la ligne à partir de laquelle commencera automatiquement l'exécution, une fois le nouveau programme chargé. Lorsque ce paramètre n'est pas spécifié, l'exécution commence à partir de la ligne ayant le plus petit numéro.
- L'option **DELETE** de l'instruction CHAIN MERGE permet de supprimer les lignes de l'ancien programme se trouvant entre les numéros spécifiés.

COMMENTAIRES

Les instructions CHAIN et CHAIN MERGE provoquent l'exécution immédiate du nouveau programme chargé. Lorsque le programme appelé par CHAIN MERGE possède certaines lignes portant le même numéro que des lignes du programme se trouvant déjà en mémoire centrale, celles de l'ancien programme sont effacées au profit de celles du nouveau. Les valeurs de variables ne sont pas effacées par l'instruction CHAIN MERGE. Par contre, les informations contenues dans les instructions DEF FN, DEFINT, DEFREAL et DEFSTR sont perdues.

Ces instructions doivent donc être réexécutées une fois la fusion réalisée. Il en va de même pour les boucles (FOR...NEXT, WHILE...WEND) et les branchements (GOSUB, ON ERROR GOTO), les pointeurs devant être repositionnés. Les fichiers doivent également être réouverts.

Les programmes protégés (c'est-à-dire ceux qui ont été sauvegardés en utilisant l'option ,P de l'instruction SAVE) ne peuvent pas être fusionnés au moyen de l'instruction CHAIN ou CHAIN MERGE.



Fonction

La fonction CHR$ renvoie le caractère correspondant au code ASCII donné comme argument à la fonction.

FORMAT

CHR$(n)

- **n** est une expression numérique entière dont la valeur est comprise entre 0 et 255.

COMMENTAIRES

Seuls les caractères dont le code ASCII est compris entre 32 et 255 peuvent être affichés à l'écran par l'intermédiaire de la fonction CHR$. Les caractères ayant un numéro de code inférieur à 32 correspondent à des fonctions BASIC particulières. Malgré tout, ces codes peuvent être donnés comme arguments à la fonction CHR$ qui les imprime, dans un programme, comme des caractères. Par exemple, l'exécution de PRINT CHR$(7) provoque l'émission d'un son.

Il est à noter que la plupart des imprimantes ne disposent pas du jeu complet de caractères correspondant à l'ensemble des codes compris entre 32 et 255. Les caractères graphiques, par exemple, ne peuvent souvent être affichés qu'à l'écran.

La fonction ASC effectue l'opération inverse de celle qui est réalisée par CHR$, en renvoyant le code ASCII d'un caractère qui lui est donné comme argument.

La liste de tous les codes ASCII est donnée en annexe du manuel de l'utilisateur livré avec l'Amstrad.

Cette commande réinitialise toutes les variables numériques à zéro et toutes les variables de chaîne à une longueur nulle.

FORMAT

CLEAR

COMMENTAIRES

L'instruction CLEAR efface les valeurs des variables mais pas le programme en cours. L'exécution de cette commande rend les tableaux non définis et clôt les fichiers. De même, toutes les informations contenues dans les instructions DEF FN, DEFINT, DEFREAL et DEFSTR sont perdues. Le mode radian est d'autre part rétabli pour toutes les fonctions trigonométriques.

La commande CLEAR étant assez destructrice, il est recommandé de ne l'utiliser qu'avec précaution et de se souvenir que l'instruction ERASE permet de n'effacer que les tableaux, sans affecter les autres données du programme (voir la description de cette instruction).



CLEAR INPUT

Instruction
(CPC 664 et CPC 6128)

L'instruction CLEAR INPUT efface le contenu de la mémoire tampon du clavier.

FORMAT

CLEAR INPUT

COMMENTAIRES

Cette instruction peut être en particulier utilisée lorsque plusieurs groupes de données doivent être saisis successivement au clavier au moyen des instructions INKEY$ et INKEY.

L'instruction CLG efface l'écran graphique et sélectionne éventuellement une nouvelle couleur de fond pour le mode graphique.

FORMAT

CLG < numéro de couleur >

- Le **numéro de couleur** doit être compris entre 0 et 15.

COMMENTAIRES

Lorsque le numéro de couleur n'est pas précisé, la valeur courante de l'encre de fond est conservée (voir l'instruction GRAPHICS PAPER). Cette couleur est aussi celle sur laquelle s'inscrivent les caractères écrits en mode graphique (voir l'instruction TAG).

Les instructions CLOSEIN et CLOSEOUT sont destinées à fermer tous les fichiers ouverts en lecture (CLOSEIN) ou en écriture (CLOSEOUT). Avant la fermeture, les données relatives aux fichiers ouverts en écriture sont transférées des mémoires tampon correspondantes. Ces emplacements mémoire sont alors libérés.

FORMAT

CLOSEIN

CLOSEOUT

COMMENTAIRES

Un fichier doit nécessairement être fermé lorsque le transfert des données est terminé. Les instructions CLOSEIN et CLOSEOUT vident en effet les mémoires tampon associées aux fichiers et l'instruction CLOSEIN place un caractère de fin de fichier (CTRL Z) à la fin des fichiers séquentiels ouverts en mode écriture (voir l'instruction OPENIN).

Il est à noter que l'instruction END ferme automatiquement tous les fichiers ouverts.

Les instructions CLOSEIN et CLOSEOUT sont illustrées dans le programme de gestion de fichier d'adresses donné dans la rubrique OPENIN et OPENOUT.

L'instruction CLS permet d'effacer l'écran ou la fenêtre d'écran précisée.

FORMAT

CLS < #numéro de canal >

COMMENTAIRES

L'instruction CLS efface la totalité de l'écran ou la fenêtre texte précisée ; elle l'initialise dans la couleur en cours du papier (voir l'instruction PAPER). Si l'instruction CLS n'est suivie d'aucun paramètre ou si le numéro de canal est zéro, le curseur est placé en haut et à gauche de l'écran.

COMMANDES AMSDOS
CPC 664 et CPC 6128

AMSDOS est le système d'exploitation du disque (SED ou DOS en anglais) résidant en mémoire morte. Il existe une alternative à l'utilisation de AMSDOS, celle consistant à utiliser le système non résident CP/M, beaucoup plus performant mais indépendant du BASIC (voir l'Introduction). Une seule commande de CP/M sera évoquée ici, la commande FORMAT. Toute opération d'écriture sur disque (sauvegarde d'un programme par exemple) nécessite en effet que le disque en question ait été au préalable formaté. Cette opération n'est possible que sous CP/M, le système AMSDOS ne disposant pas de la commande équivalente.

AMSDOS possède deux types de commandes, les commandes externes et les commandes internes. Les commandes internes ne se différencient pas des commandes ou instructions BASIC. Elles sont donc traitées à ce titre dans le reste de l'ouvrage. Lorsque l'on utilise par exemple l'instruction PRINT #1, on utilise une instruction BASIC. Par contre, PRINT #9 est une commande AMSDOS. Cette distinction est naturellement accessoire pour l'utilisateur.

Par contre, les commandes externes de AMSDOS nécessitent la mise en œuvre d'une procédure d'appel légèrement différente. Elles sont accessibles à partir de BASIC en tapant le signe @, ce que l'on obtient en pressant simultanément la touche SHIFT et la touche portant les deux symboles \ et @.

Avant de passer en revue la liste des commandes externes de AMSDOS, il est nécessaire de revenir sur la notion de nom de fichier (fichier programme, fichier de données, etc.). D'une manière générale, un nom de fichier, sous AMSDOS, est formé de trois parties :

- Le nom d'unité, A: ou B:. Cela ne concerne que les utilisateurs disposant de deux unités de disques. Chaque unité est alors identifiée par une lettre (suivie de :) par le système d'exploitation. L'unité intégrée au clavier est l'unité A:, qui est aussi, sauf spécification contraire, l'unité par défaut. Lorsque l'on effectue une opération mettant en jeu l'unité par défaut, le nom d'unité n'a pas besoin d'être mentionné.

- La partie principale du nom de fichier. C'est cette partie qui est communément appelée "nom de fichier" dans le reste de l'ouvrage. La partie principale peut comporter de 1 à 8 caractères, sans espace blanc intermédiaire.
- L'extension de nom du fichier, formée d'un point suivi de 1 à 3 caractères (sans espace blanc intermédiaire). L'extension est facultative. Lorsqu'elle n'est pas spécifiée par l'utilisateur au moment d'une sauvegarde, le système d'exploitation AMSDOS ajoute lui-même une extension de nom adaptée. AMSDOS se sert en particulier de ces extensions pour établir une préséance dans la recherche des noms de fichiers.

L'extension de nom mise automatiquement par le système au moment de la sauvegarde est une extension par défaut. Si l'on utilise par exemple l'instruction :

```
SAVE "BLANCHE"
```

pour sauvegarder le fichier programme susnommé, AMSDOS transformera automatiquement le nom BLANCHE en BLANCHE.BAS. Il est par contre tout à fait loisible à l'utilisateur de sauvegarder le fichier en lui donnant une extension différente, .T par exemple, au moyen de l'instruction :

```
SAVE "BLANCHE.T"
```

Les extensions de noms utilisées par défaut par AMSDOS sont les suivantes :

. Absence de spécification autre que le point. C'est par exemple le cas pour un fichier de données créé au moyen d'une instruction :

```
OPENOUT "MONFRIC"
```

.BAS Programme BASIC sauvegardé par une instruction :
SAVE "MONFRIC" ou SAVE "MONFRIC",P ou
SAVE "MONFRIC",A.

.BIN Fichier binaire sauvegardé au moyen d'une instruction :
SAVE "MONFRIC",B,paramètres binaires.

.BAK Fichier de sauvegarde. Ce fichier est automatiquement constitué par AMSDOS lorsque l'on sauvegarde un fichier plusieurs fois sous un même nom (lorsqu'on lui a par exemple apporté certaines modifications). Si l'on sauvegarde le fichier modifié "Gencébas", l'ancienne version sera conservée sur la disquette sous le nom :

"Gencébas.BAK"

UTILISATION DE CARACTÈRES GÉNÉRIQUES

Ces caractères sont essentiellement utilisés pour établir un répertoire partiel de la disquette ou pour effacer ou copier une classe particulière de fichiers. Il existe deux caractères génériques, * et ?.

* Remplace un groupe de caractères. Par exemple, *.BAS désigne tous les fichiers ayant l'extension .BAS, quelle que soit la partie principale du nom de ces fichiers. De la même manière, FIC*.BAS désigne tous les fichiers ayant l'extension .BAS et dont la partie principale du nom commence par les trois lettres F, I et C.

? Remplace un caractère particulier. En reprenant l'exemple précédent, FIC?.BAS désignera aussi bien le fichier FIC7.BAS que FIC8.BAS, mais pas le fichier FIC10.BAS (qui sera par contre reconnu par FIC*.BAS).

COMMANDES EXTERNES DE AMSDOS

¦A, ¦B et ¦DRIVE

¦A sélectionne l'unité A comme unité par défaut, ¦B opérant de la même manière avec l'unité B.

¦DRIVE,"A" est équivalent à ¦A et ¦DRIVE,"B" est équivalent à ¦B.

¦CPM

Provoque le passage sous CP/M. Cette commande ne doit être utilisée qu'après sauvegarde d'un éventuel fichier se trouvant en mémoire centrale. Dans le cas contraire, ce fichier serait perdu.

¦DIR < ,chaîne de caractères >

Établit le répertoire de la disquette. La chaîne de caractères (optionnelle) sert à établir un répertoire partiel de la disquette, en utilisant les caractères génériques, comme dans l'exemple :

```
¦DIR,"FIC.*"
```

Lorsque aucune chaîne n'est spécifiée, le répertoire complet est affiché et la commande ¦DIR s'apparente alors à la commande interne CAT.

¦DISC, ¦DISC.IN et ¦DISC.OUT

Définit l'unité de disque comme unité standard d'entrée (¦DISC.IN) ou comme unité standard de sortie (¦DISC.OUT) ou comme unité standard d'entrée et de sortie (¦DISC).
Ces commandes n'ont d'intérêt que lorsqu'une unité de disque et un magnétocassette sont utilisés simultanément.

¦ERA,chaîne de caractères

Efface de la disquette les fichiers non protégés dont le nom (complet) est spécifié par la chaîne de caractères. Les caractères génériques peuvent être employés dans cette chaîne. Par exemple :

```
¦ERA,"*.BAK"
```

efface tous les fichiers de sauvegarde.
La commande ¦ERA,"*.*" ne peut s'utiliser que dans trois circonstances :

- vous disposez d'une ou plusieurs copies de votre disquette ;
- vous êtes atteint de masochisme pervers ;
- vous n'avez pas tout à fait compris la signification du caractère *.

¦REN,nouveau-nom,ancien-nom

Permet de modifier le nom d'un fichier stocké sur disquette. Les caractères génériques ne sont pas autorisés dans cette commande.

¦TAPE , ¦TAPE.IN et ¦TAPE.OUT

Définit le magnétocassette comme unité standard d'entrée (¦TAPE.IN) ou comme unité standard de sortie (¦TAPE.OUT), ou encore comme unité standard d'entrée et de sortie (¦TAPE).
Ces commandes n'ont d'intérêt que lorsqu'un magnétocassette et une unité de disque sont utilisés simultanément.

¦USER,expression numérique entière

Permet d'effectuer une partition de la disquette. L'expression numérique doit être comprise entre 0 et 15. Les commandes d'accès disque telles que CAT, LOAD, SAVE, DIR, etc. s'appliquent toujours à la partition définie par la commande ¦USER en cours.

LA COMMANDE FORMAT de CP/M

Le formatage est une opération qui doit être effectuée sur toute disquette vierge, avant que celle-ci ne puisse être utilisée pour sauvegarder un programme ou des données. Par contre, il ne faut jamais reformater une disquette contenant des informations déjà enregistrées et que l'on souhaite conserver. Le formatage a en effet pour conséquence d'effacer toute information se trouvant déjà sur la disquette.

La procédure à suivre pour formater une disquette est la suivante (système monodisque) :

- Taper ¦CPM (passage sous le système CP/M).
- Placer la disquette système CP/M dans le lecteur.
- Taper FORMAT. Après chargement de l'utilitaire de formatage, l'écran affiche :

  "Please insert disc to be formatted into drive A then press any key"

- Placer le disque à formater dans le lecteur puis taper une touche quelconque. Une fois l'opération terminée, un message demande si une autre disquette doit être formatée. Pour répondre négativement, taper N.

- Placer, en réponse au nouveau message affiché, la disquette système CP/M dans le lecteur, puis taper une touche quelconque.
- Retourner sous BASIC en tapant AMSDOS.



Commande

La commande CONT reprend l'exécution du programme après une interruption.

FORMAT

CONT

COMMENTAIRES

Cette commande est généralement utilisée en conjonction avec l'instruction STOP écrite dans un programme. Cette dernière suspend en un point précis l'exécution du programme, permettant par exemple à l'utilisateur d'examiner et/ou de modifier en mode direct la valeur de certaines variables. L'exécution du programme peut alors être reprise, à partir de l'endroit même où s'est produite l'interruption, au moyen de la commande CONT. Si l'on souhaite reprendre l'exécution en un autre point du programme, il convient d'utiliser l'instruction GOTO en mode direct et non la commande CONT. De même, CONT ne peut pas être utilisée lorsque le programme (et non la valeur des variables) a été modifié au cours de l'interruption.

La commande CONT est également utilisable pour reprendre l'exécution d'un programme qui a été interrompu après :

- l'apparition d'une erreur ;
- l'exécution, par le programme, d'une instruction END ;
- la frappe deux fois de suite de la touche ESC. Cela a pour effet de suspendre l'exécution du programme, le message "Break in xxx" (interruption en xxx) étant alors affiché. xxx est le numéro de ligne de l'instruction en cours d'exécution au moment de l'interruption.



La fonction COPYCHR$ assigne, à la variable alphanumérique à laquelle elle est associée, le caractère, dans le canal indiqué, placé à la position courante du curseur.

FORMAT

COPYCHR$(# numéro de canal)

COMMENTAIRES

Cette fonction permet d'attribuer à une variable alphanumérique un caractère placé à n'importe quel endroit de l'écran.

Le programme donné en illustration de l'instruction KEY utilise la fonction COPYCHR$ pour lire les caractères écrits dans la matrice du générateur.



CURSOR

Instruction (CPC 664 et CPC 6128)

L'instruction CURSOR permet d'afficher le curseur sur l'écran lors de l'exécution des commandes d'entrée clavier (INKEY$ et INPUT).

FORMAT

CURSOR < paramètre système > < , paramètre utilisateur >

- Les **paramètres** ont pour valeur 0 ou 1 ; ils correspondent respectivement à "affiché" et "non affiché".

Exemples

`CURSOR 0,0` et `CURSOR 1,0`

Le curseur n'est pas affiché sur l'écran lors de l'exécution des commandes INKEY$ ou INPUT.

```
CURSOR 0,1
```

Le curseur est affiché sur l'écran seulement lors de l'exécution de la commande INPUT.

```
CURSOR 1,1
```

Le curseur est affiché sur l'écran lors de l'exécution des commandes INKEY$ ou INPUT.

COMMENTAIRES

Les deux paramètres ne sont pas nécessaires : il suffit d'en indiquer un. Dans ce cas, l'autre conserve la valeur qu'il avait précédemment.

A l'initialisation du système, les valeurs par défaut sont 0 et 1 (CURSOR 0,1).

EXEMPLE DE PROGRAMME

Le programme ci-dessous illustre l'état du curseur en fonction des paramètres associés à l'instruction CURSOR.

```
10 CLS
20 FOR I=0 TO 1
30 FOR J=0 TO 1
40 CURSOR I,J
50 PRINT "CURSOR";I;",";J
60 PRINT "                        INKEY $ ---> "
;
70 C$=""
80 WHILE C$="" : C$=INKEY$ : WEND
90 PRINT C$
110 INPUT "                       INPUT    --->
",C$
120 PRINT
130 PRINT
140 NEXT J
150 NEXT I
```

La fonction DEC$ convertit une valeur numérique en une valeur alphanumérique, suivant un format donné.

FORMAT

DEC$(expression numérique,spécification du format)

- Sur le CPC 464, une erreur dans l'interpréteur BASIC nécessite d'employer une syntaxe légèrement différente, en mettant deux parenthèses ouvrantes après le mot clé **DEC$** et une seule parenthèse fermante après l'argument **spécification du format**.
- Les caractères destinés à établir le format sont les suivants :

#	Caractérise un champ numérique à un chiffre.
.	Spécifie un point décimal.
,	Place une virgule entre les groupes de trois chiffres situés à gauche du point décimal.
+	Génère un signe plus (+) pour les nombres positifs ou un signe moins (–) pour les nombres négatifs.
–	Placé à la fin du format, génère un signe moins (–) après les nombres négatifs.
**	Place un astérisque au début du nombre et, si celui-ci est précédé d'espaces, les remplace par des astérisques.
££	Place le caractère livre (£) au début du nombre.
$$	Place le caractère dollar ($) au début du nombre.
**£	Place le caractère livre (£) au début du nombre et, si celui-ci est précédé d'espaces, les remplace par des astérisques.
**$	Place le caractère dollar ($) au début du nombre et, si celui-ci est précédé d'espaces, les remplace par des astérisques.

^^^^ Précédé de caractères dièses (#), spécifie une notation exponentielle.

COMMENTAIRES

Cette instruction a une mise en œuvre analogue à celle de la fonction PRINT USING ; le résultat, au lieu d'être affiché sur l'écran, est assigné à une variable alphanumérique.

EXEMPLE DE PROGRAMME

Le programme ci-dessous illustre l'utilisation de l'instruction DEC$; différents formats sont établis et, lors de l'affichage des résultats, le nombre est précédé de la structure de son format.

```
10 CLS
20 PRINT "TAPEZ UN NOMBRE QUELCONQUE :"
30 INPUT N
40 PRINT
50 PRINT "      FORMAT";TAB(22)" RESULTAT"
60 PRINT
70 A$=DEC$(N,"#######")
80 PRINT "#######              :";TAB(22)A$
90 PRINT
100 A$=DEC$(N,"#######.###")
110 PRINT "#######.###          :";TAB(22)A$

120 PRINT
130 A$=DEC$(N,"#######,.###")
140 PRINT "#######,.###         :";TAB(22)A$
150 PRINT
160 A$=DEC$(N,"**#######,.###")
170 PRINT "**#######,.###       :";TAB(22)A$

180 PRINT
190 A$=DEC$(N,"#########,.###")
200 PRINT "#########,.###       :";TAB(22)A$

210 PRINT
220 A$=DEC$(N,"$$#######,.###")
230 PRINT "$$#######,.###       :";TAB(22)A$
240 PRINT
250 A$=DEC$(N,"**########,.###")
260 PRINT "**########,.###      :";TAB(22)A$
270 PRINT
```

```
280 A$=DEC$(N,"**$########,.###")
290 PRINT "**$########,.###   :";TAB(22)A$
300 PRINT
310 A$=DEC$(N,"+########,.###")
320 PRINT "+########,.###     :";TAB(22)A$
330 PRINT
340 A$=DEC$(N,"########,.###-")
350 PRINT "########,.###-     :";TAB(22)A$
360 PRINT
370 A$=DEC$(N,"#.#####^^^^")
380 PRINT "#.#####^^^^        :";TAB(22)A$

390 PRINT
400 A$=DEC$(N,"#.#####^^^^+")
410 PRINT "#.#####^^^^+       :";TAB(22)A$
420 GOTO 420
```

Dans l'exemple de résultats donné ci-dessous, la valeur entrée est le nombre 32.1.

FORMAT		RESULTAT
########	:	32
########.###	:	32.100
########,.###	:	32.100
**#########,.###	:	*********32.100
££#########,.###	:	£32.100
$$#########,.###	:	$32.100
**£#########,.###	:	*********£32.100
**$#########,.###	:	*********$32.100
+########,.###	:	+32.100
########,.###-	:	32.100
#.#####↑↑↑↑	:	.32100E+02
#.#####↑↑↑↑+	:	3.21000E+01+

Instruction (DEF FN) et fonction (FN)

L'instruction DEF FN déclare une fonction définie par l'utilisateur et FN permet de calculer la valeur de cette fonction pour la liste de paramètres spécifiés.

FORMATS

DEF FN nom de fonction < (liste de variables) > =expression

FN nom de fonction < (liste de paramètres) >

- Le **nom de fonction** peut être un nom de variable numérique ou une chaîne de caractères. Ce nom sert à identifier la fonction.
- La **liste de variables** est une suite de variables numériques ou de variables de chaîne, séparées par des virgules. Ces noms sont fictifs ; ils peuvent être remplacés par d'autres noms lors de l'appel de la fonction au moyen de FN.
- L'**expression** est la relation de définition de la fonction. Il doit y avoir correspondance entre le type de l'expression (numérique ou fonction de chaîne) et le nom déclaré pour la fonction.
- La **liste des paramètres** est une suite de valeurs qui sont passées aux variables, dans l'ordre où elles sont écrites dans l'instruction de déclaration de la fonction. L'expression sera calculée en utilisant ces valeurs. Il doit y avoir correspondance entre le type du paramètre et celui de la variable qui le reçoit.

COMMENTAIRES

L'instruction DEF FN ne peut être utilisée qu'en mode programme. Elle doit toujours être exécutée avant que la fonction ainsi définie ne puisse être appelée au moyen de FN.

La liste de variables peut comporter des noms de variables utilisés pour d'autres fonctions du programme. Les valeurs de celles-ci ne

seront pas affectées par les paramètres de FN passés aux variables de DEF FN. Réciproquement, il n'est pas nécessaire de spécifier le nom d'une variable comme argument de DEF FN si la valeur courante de cette variable doit être utilisée par la fonction définie. Le second exemple donné ci-dessous illustre cette possibilité.

EXEMPLES

1. Définition d'une fonction SURF calculant la surface d'un cercle :

```
10 DEF FNSURF(R) = PI*R*R
20 INPUT "Rayon";X
30 PRINT "Surface du cercle =";FNSURF(X)
```

2. Calcul de la racine cubique d'un nombre :

```
10 DEF FNRacine = EXP(1/3*LOG(X))
20 INPUT "Nombre dont la racine cubique doit être calculée";X
30 PRINT "Racine cubique de ce nombre";FNRacine
```

Les instructions DEFINT, DEFREAL et DEFSTR déclarent les variables spécifiées dans le type entier (DEFINT), réel (DEFREAL) ou chaîne de caractères (DEFSTR).

FORMAT

DEFtype < – lettre > < ,lettre > < lettre >

- **type** est l'un des suffixes INT, REAL ou STR.
- **lettre** est une lettre de l'alphabet.

Exemples

```
DEFINT R-V
```

déclare que toutes les variables dont le nom commence par la lettre R, S, T, U ou V sont de type entier.

```
DEFSTR A
```

déclare que toutes les variables dont le nom commence par la lettre A sont de type chaîne de caractères.

```
DEFREAL C-E,K-M
```

déclare que toutes les variables dont le nom commence par les lettres C, D, E, K, L et M sont de type numérique réel.

RÈGLES DE SYNTAXE ET COMMENTAIRES

L'instruction DEFtype doit être exécutée avant que la variable ainsi déclarée ne soit utilisée dans le programme. Il est donc préférable de placer ces instructions dans les premières lignes du programme. Toutes les variables numériques non déclarées sont considérées

comme étant des variables numériques réelles. Il est également possible de déclarer le type d'une variable en faisant suivre son nom d'un des symboles % et ! (ou $ pour une variable de chaîne). Ces symboles de déclaration ont toujours la priorité sur une instruction DEFtype, c'est-à-dire qu'une variable A initialement déclarée comme nombre entier par l'instruction DEFINT A pourra être redéfinie, à un autre endroit du programme, comme étant une variable dont la valeur est exprimée par un nombre réel (en écrivant par exemple A! = A).

Les nombres entiers sont codés sur deux octets. Leur valeur doit être comprise entre −32768 et 32767. Les nombres réels peuvent prendre toute valeur comprise entre 2.9E−39 et 1.7E+38. Ces valeurs sont codées sur cinq octets (voir la fonction @).

L'instruction DEG positionne le mode *degré* comme unité de calcul des fonctions trigonométriques (ATN, COS, SIN, TAN). L'instruction RAD positionne le mode *radian* comme unité de calcul pour les mêmes fonctions.

FORMATS

DEG

RAD

COMMENTAIRES

Le mode radian est établi par défaut lors de la mise sous tension de l'ordinateur. Lorsque le mode degré est sélectionné au moyen de l'instruction DEG, ce mode est conservé jusqu'à ce que le système soit réinitialisé ou qu'une commande NEW, CLEAR, LOAD, RUN, etc. (ou une instruction RAD) soit exécutée.



DELETE

Commande

La commande DELETE supprime certaines lignes de programme présentes en mémoire centrale de l'ordinateur.

FORMAT

DELETE < ligne 1 > < –ligne 2 >

- **ligne 1** est le numéro de la première ligne à supprimer.
- **ligne 2** est le numéro de la dernière ligne à supprimer.

COMMENTAIRES

Après exécution de la commande DELETE, l'ordinateur revient en mode direct. Il en résulte que si une seule ligne doit être effacée, il est plus simple de taper, en mode programme, le numéro correspondant.

```
DELETE ligne
```

n'efface que la ligne dont le numéro est spécifié.

```
DELETE ligne 1 - ligne 2
```

efface toutes les lignes dont les numéros sont compris entre ceux spécifiés dans la commande. Par exemple :

```
DELETE 20-60
```

supprime toutes les lignes de programme comprises entre les numéros 20 et 60 (inclus).

```
DELETE - ligne 2
```

efface toutes les lignes comprises entre la première ligne de programme et celle dont le numéro est spécifié. Par exemple :

```
DELETE -200
```

efface toutes les lignes de programme dont le numéro est compris entre 0 et 200 (ligne 200 comprise).

```
DELETE
```

(sans numéro de ligne) : efface la totalité du programme se trouvant en mémoire centrale.



DERR

Fonction (CPC 664 et CPC 6128)

La fonction DERR retourne un code d'erreur dans le cas où une erreur s'est produite au cours d'un accès disque.

FORMAT

DERR

COMMENTAIRES

Comme tous les autres codes d'erreur, ceux qui sont liés aux accès disque portent un numéro retourné par la fonction ERR (voir la rubrique ''Gestion des erreurs''). Il s'agit en l'occurrence des numéros 31 et 32. Lorsqu'une telle erreur se produit, la fonction DERR permet de rechercher plus précisément l'origine de l'erreur. Elle lit un indicateur codé sur un octet. Le bit de poids fort (bit 7) de cet octet est positionné si l'erreur a été décelée par le système d'exploitation AMSDOS. Il vaut au contraire 0 si l'erreur a été indiquée par le contrôleur du disque. Dans ce dernier cas, le bit 6 de l'indicateur est positionné. En conséquence, la fonction DERR retournera un numéro de code supérieur à 128 dans le cas d'une erreur AMSDOS et un numéro compris entre 64 et 128 dans le cas d'une erreur détectée par le contrôleur.

Par exemple, la valeur 148 correspond à une tentative d'écriture sur un disque plein. Il s'agit d'une erreur décelée par le système d'exploitation. La valeur 72 est retournée par la fonction DERR lorsqu'un accès disque est demandé alors qu'il n'y a pas de disquette dans l'unité spécifiée. Cette erreur est détectée par le contrôleur. La liste complète des codes d'erreur au cours d'un accès disque est donnée dans le guide de l'utilisateur livré avec l'Amstrad.

Les instructions DI et EI sont destinées respectivement à interdire et à autoriser les interruptions d'une partie de programme ou de sous-programme.

FORMATS

DI

EI

COMMENTAIRES

Les ordinateurs Amstrad possèdent une horloge qui permet d'utiliser simultanément quatre chronomètres. En fonction du laps de temps écoulé, des sous-programmes peuvent être exécutés. Si l'exécution de certaines parties de programme ne doit pas être interrompue, on utilise les instructions DI (*Disable Interrupt*, annulation d'une interruption) et EI (*Enable Interrupt*, rétablissement d'une interruption) pour délimiter ces zones (voir l'instruction EVERY...GOSUB).

L'instruction DI marque le début de la zone ; celle-ci se termine après une instruction EI ou après une instruction RETURN dans le cas d'un sous-programme d'interruption associé à une instruction EVERY...GOSUB ou AFTER...GOSUB. Lorsqu'une interruption est annulée, l'appel du sous-programme est seulement retardé (et non pas supprimé). Le branchement s'effectuera dès que l'instruction DI aura été désactivée (et que les autres conditions de branchement seront réalisées).

Les interruptions de programme générées par des erreurs ou par la frappe de la touche Esc ne sont pas affectées par l'instruction DI. Les interruptions évoquées ici sont naturellement les interruptions logicielles. Les interruptions du système (qui se produisent au moins toutes les 20 millisecondes) ne sont en rien affectées par les instructions DI et EI.



Instruction

L'instruction DIM permet de spécifier les dimensions maximales de tableaux de variables et réserve la place mémoire en conséquence.

FORMAT

DIM variable(indices) < ,variable(indices) > ...

- **variable** est le nom désignant le tableau ; chacune de ces variables aura pour nom celui du tableau suivi, entre parenthèses, de l'indice particulier à cette variable.
- **indices** est une liste d'expressions numériques, séparées par une virgule, qui définissent les dimensions du tableau.

Exemples

```
DIM T(15)
```

définit un tableau à une dimension ayant 16 variables numériques : T(0), T(1), T(2), ..., T(16).

```
DIM A(20,30)
```

définit un tableau à deux dimensions ayant 21×31 éléments ou variables numériques.

```
DIM A1(15),A2(20)
```

définit deux tableaux à une dimension, A1 et A2, ayant respectivement 16 et 21 éléments.

```
DIM A$(30)
```

définit un tableau à une dimension ayant 31 variables de chaîne comme éléments.

COMMENTAIRES

Il est possible d'utiliser dans un programme des variables de tableau (ou variables indicées) qui n'ont pas été définies au préalable par une instruction DIM, à condition que les indices correspondants n'excèdent pas 10. Lorsque l'indice d'une variable appelée dépasse celui qui est spécifié dans l'instruction DIM, un message d'erreur "Subscript out of range in xxx" (indice hors limites à la ligne xxx) est affiché.

Une seule instruction DIM peut être utilisée pour dimensionner plusieurs tableaux, comme dans le troisième exemple donné ci-dessus. Il ne s'agit cependant pas d'une obligation ; un programme peut comporter autant d'instructions DIM qu'on le souhaite.

L'indice de la première variable d'un tableau est toujours égal à 0. Le nombre maximal de dimensions possibles pour un tableau ainsi que le nombre maximal d'éléments par dimension sont limités par la taille de la mémoire disponible et par la longueur du programme utilisant ce tableau. Le programme qui illustre l'instruction ERASE montre comment calculer la place mémoire disponible ou celle qui est occupée par un tableau. Cette instruction permet également de redimensionner plusieurs fois un même tableau dans un programme.

Tant qu'une valeur n'a pas été assignée à une variable d'un tableau dimensionné, celle-ci est égale à 0 (valeur par défaut), qu'il s'agisse d'une variable numérique ou d'une variable de chaîne.

Les instructions DRAW et DRAWR permettent de tracer une ligne sur l'écran à partir de la position courante du curseur graphique. L'instruction DRAW est associée à des coordonnées absolues, tandis que l'instruction DRAWR est associée à des coordonnées relatives.

FORMATS

DRAW coordonnée x, coordonnée y < ,numéro d'encre >
< ,mode d'encre >

DRAWR déplacement x, déplacement y < ,numéro d'encre >
< ,mode d'encre >

- Le **numéro d'encre** peut prendre une valeur comprise entre 0 et 15 en fonction de l'encre choisie.
- Le **mode d'encre** est un paramètre qui n'existe que sur le CPC 664 et le CPC 6128 ; il peut prendre les valeurs 0, 1, 2, 3 ou 4 suivant le mode d'interaction avec les couleurs de l'écran graphique.

COMMENTAIRES

Les instructions DRAW et DRAWR offrent au programmeur une multitude de possibilités graphiques.

Sur les modèles CPC 664 et 6128, il est possible de réaliser une opération logique (paramètre mode d'encre) entre la couleur du stylo ou du papier en cours et l'ancienne couleur dans laquelle était affiché le pixel. Ces opérations sont les suivantes :

Valeur du paramètre	Opération
0	Pas d'opération
1	OU exclusif (XOR)
2	ET (AND)
3	OU (OR)

La couleur dans laquelle sera affiché le point concerné sur l'écran graphique dépendra du résultat de l'opération logique réalisée sur les deux valeurs. Rappelons que les opérations logiques sont effectuées bit à bit, sur les octets correspondants. Supposons par exemple qu'un pixel soit originellement affiché dans la couleur 8 et qu'il soit affecté par une instruction DRAW de paramètres 2 pour le numéro d'encre et 1 pour le mode d'encre. Pour connaître la couleur dans laquelle s'affichera le pixel, l'opération binaire suivante doit donc être effectuée :

00001000 XOR 00000010

Le résultat de cette opération (voir la rubrique ''Opérateurs logiques'') est 00001010, nombre binaire correspondant à la couleur 10. Supposons que l'instruction DRAW soit exécutée une seconde fois sur ce même point, avec la même valeur des paramètres numéro et mode d'encre. L'opération logique réalisée sera donc la suivante :

00001010 XOR 00000010

Le résultat est 00001000, ce qui correspond à la couleur 8. Le point se réaffichera donc dans la couleur qui était la sienne avant l'exécution des deux instructions DRAW. Cette propriété est tout à fait typique de l'opérateur XOR qui peut être ainsi utilisé, avec les instructions graphiques qui le permettent, pour ''allumer'' ou ''éteindre'' un point.

EXEMPLE DE PROGRAMME

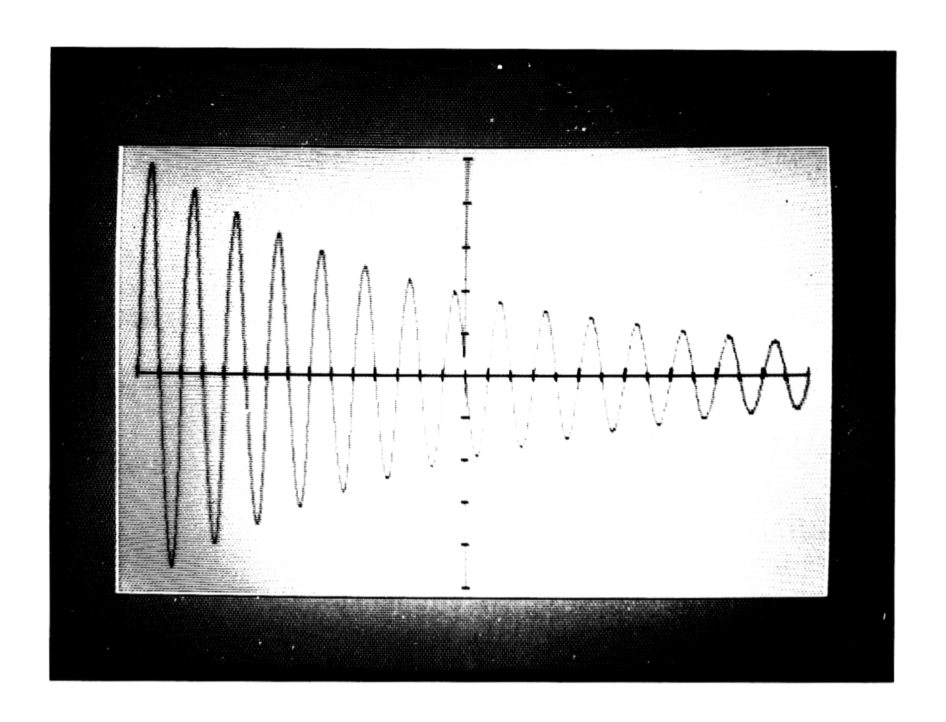
Le programme ci-dessous trace sur l'écran le graphe de la fonction Y = EXP(aX) * SIN(X) + b dans un repère orthonormé (les axes X et Y sont perpendiculaires).

Ce programme est divisé en quatre parties correspondant respectivement à l'initialisation et au tracé du cadre (lignes 10 à 100), au tracé des axes (lignes 110 à 150), à la mise en place de la graduation (lignes 160 à 280) puis au tracé de la fonction (lignes 290 à 350).

```
10 CLS
20 MODE 2
30 BORDER 3
40 ORIGIN 0,0
48 '
```

→

```
49 '***********************
50 ' Trace du cadre
51 '***********************
60 MOVE 0,0
70 DRAW 639,0
80 DRAW 639,399
90 DRAW 0,399
100 DRAW 0,0
108 '
109 '***********************
110 ' Trace des axes
111 '***********************
120 MOVE 19,199
130 DRAWR 600,0
140 MOVE 319,9
150 DRAWR 0,379
158 '
159 '***********************
160 ' Trace des graduations
161 '***********************
170 MOVE 19,199
180 FOR I=19 TO 619 STEP 20
190 MOVE I,199
200 MOVER 0,5
210 DRAWR 0,-10
220 NEXT I
230 MOVE 319,9
240 FOR I=9 TO 389 STEP 38
250 MOVE 319,I
260 MOVER -3,0
270 DRAWR  6,0
280 NEXT I
290 ' Trace de y=sin(x)
300 MOVE 19,199
310 X=0
320 FOR I=0 TO 30*PI STEP 5*PI/100
330 X=X+1
340 DRAW X+19,(EXP(-0.02*I)*SIN(I)*190)+(189+9)
350 NEXT I
358 '
359 '***********************
360 ' Boucle d'attente
361 '***********************
370 C$=""
380 WHILE C$="" : C$=INKEY$ : WEND
390 MODE 1
400 END
```





Commande

La commande EDIT permet d'afficher une ligne quelconque de programme sur l'écran puis de la modifier à l'aide des commandes d'édition (touches flèche, touches Clr et Del, ...).

FORMAT

EDIT numéro de ligne

COMMENTAIRES

Il existe plusieurs moyens d'éditer une ligne de programme (voir en particulier la commande LIST). Le Chapitre 2 contient un paragraphe présentant les possibilités d'édition offertes par l'Amstrad.

L'instruction END met fin à l'exécution du programme et renvoie en mode direct après fermeture de tous les fichiers qui ont été ouverts au cours de l'exécution du programme.

FORMAT

END

COMMENTAIRES

Un programme peut comporter autant d'instructions END qu'on le désire, certaines pouvant être exécutées de manière conditionnelle (comme clause d'une instruction IF...THEN...ELSE par exemple). A la différence de l'instruction STOP, END ferme tous les fichiers et ne génère pas de message d'interruption. L'instruction END, en tant que dernière instruction à exécuter dans un programme, est facultative.



EOF

Fonction

La fonction EOF permet de tester si la fin d'un fichier (ouvert sur cassette ou sur disque) est atteinte.

FORMAT

x = EOF

COMMENTAIRES

La fonction EOF retourne la valeur −1 (vrai) si aucun fichier n'est ouvert ou si la fin du fichier est atteinte lors d'une lecture au moyen des instructions INPUT# et LINE INPUT#. Si le caractère de fin de fichier (1A) n'est pas rencontré au cours de la lecture, la valeur 0 est retournée par la fonction.

EXEMPLE DE PROGRAMME

Le programme de ''carnet d'adresses'' qui illustre les commandes OPENIN et OPENOUT utilise la fonction EOF lors de la lecture du fichier ''AGENDA''. Tant que la fin du fichier n'est pas atteinte (WHILE NOT EOF), la valeur de INC% est incrémentée de 1 puis une chaîne de caractères est lue et assignée à la variable T$(INC%). Lorsque le caractère de fin de fichier est rencontré, la fonction EOF retourne la valeur −1 et l'exécution de la boucle de lecture prend fin.

L'instruction ERASE efface de la mémoire centrale de l'ordinateur les tableaux spécifiés comme paramètres de cette instruction.

FORMAT

ERASE nom de tableau < ,nom de tableau >

- **nom de tableau** désigne les noms des tableaux qui doivent être effacés et qui ont été déclarés auparavant dans une instruction DIM.

Exemples

```
ERASE A
```

efface le tableau A ;

```
ERASE A,B
```

efface les tableaux A et B.

COMMENTAIRES

Cette instruction permet de libérer de la place en mémoire centrale de l'ordinateur si certains tableaux sont devenus inutiles. Lorsqu'un tableau a été ainsi effacé, il est toujours possible de le redimensionner plus loin dans le programme (avec une instruction DIM). L'effacement est même un préalable indispensable à toute opération de redimensionnement.

EXEMPLE DE PROGRAMME

Ce programme a pour objet de calculer la place occupée en mémoire centrale de l'ordinateur par des tableaux de valeurs numériques. Ces tableaux ont différentes tailles et différentes formes. L'ins-

truction ERASE est utilisée ici en conjonction avec la fonction FRE qui renvoie, lorsqu'elle est appelée, la place mémoire restant disponible (voir la description de cette fonction).

Trois tableaux T1(I), T2(J) et T3(K) ayant chacun une seule dimension mais un nombre d'éléments différent (11, 101 et 1001, respectivement) sont tout d'abord dimensionnés (ligne 20). Le sous-programme de calcul qui commence à la ligne 150 et qui se termine à la ligne 210 est ensuite appelé une première fois (ligne 70). Ce sous-programme calcule la place mémoire disponible (en nombre d'octets) avant et après l'effacement d'un tableau au moyen de l'instruction ERASE ; la différence est affichée. A la ligne 210, les trois tableaux ont été effacés de sorte qu'il est possible de les redimensionner, lors du retour au programme principal (ligne 90). La même opération est ensuite recommencée pour trois autres tableaux à deux dimensions (lignes 90 à 130).

```
10 CLS
20 DIM T1(10),T2(100),T3(1000)
30 A$(1)="10"
40 A$(2)="100"
50 A$(3)="1000"
60 PRINT "Dimension","Nombre d'octets oc
cupes"
65 PRINT : PRINT
70 GOSUB 150
80 PRINT
90 DIM T1(10,10),T2(10,100),T3(5,1000)
100 A$(1)="10 * 10"
110 A$(2)="10 * 100"
120 A$(3)=" 5 * 1000"
130 GOSUB 150
140 END
150 A1=FRE(1) : ERASE T1
160 A2=FRE(1) : PRINT A$(1),A2-A1
170 A2=FRE(1) : ERASE T2
180 A3=FRE(1) : PRINT A$(2),A3-A2
190 A3=FRE(1) : ERASE T3
200 A4=FRE(1) : PRINT A$(3),A4-A3
210 RETURN
```

L'exécution de ce programme permet plusieurs constatations :

- La place mémoire réservée pour des variables dimensionnées est occupée, que les tableaux correspondants soient ou non remplis

(dans cet exemple ils ne le sont pas, de sorte que tous les éléments sont initialisés à zéro).

- Chaque élément de tableau (numérique) est un nombre stocké sur 5 octets.
- Le premier indice d'un tableau est toujours 0. Ainsi, un tableau à une seule dimension du type T2(100) aura 101 éléments : de même, un tableau à deux dimensions du type T3(5,1000) aura 6 × 1001 = 6006 éléments.
- Les tableaux à deux dimensions requièrent 12 octets de plus par tableau.

Un tel calcul peut être utile afin d'estimer le nombre de valeurs numériques pouvant être stockées en mémoire centrale, en fonction de la configuration du système.

EVERY...GOSUB

Instruction

L'instruction EVERY...GOSUB provoque un branchement vers un sous-programme à intervalles de temps réguliers.

FORMAT

EVERY n1 < ,n2 > GOSUB numéro de ligne

- **n1** est un nombre entier spécifiant l'intervalle de temps au bout duquel le sous-programme associé doit être exécuté. L'unité de temps définie est égale à 0,02 seconde (1/50) ; la valeur 50 correspond donc à 1 seconde.
- **n2** est un nombre entier caractérisant le chronomètre associé à l'instruction. L'Amstrad possède quatre chronomètres numérotés de 0 à 3. Si n2 n'est pas spécifié, la valeur prise par défaut est 0.
- Le **numéro de ligne** est celui de la première ligne du sous-programme à exécuter à intervalles de temps réguliers. Le sous-programme doit se terminer par une instruction RETURN.

COMMENTAIRES

Un sous-programme peut être associé à chacun des quatre chronomètres ; leur exécution est obtenue lorsque les quatre conditions suivantes sont simultanément réalisées :

- Le temps spécifié dans l'instruction EVERY...GOSUB est écoulé.
- Il n'y a pas de branchement en attente spécifié par un chronomètre ayant un ordre de priorité plus élevé. L'ordre de priorité des chronomètres est le suivant : 3, 2, 1, 0.
- Il n'y a pas d'instruction en cours d'exécution ; sinon il faudra attendre qu'elle soit terminée pour que le sous-programme puisse être lancé. Cela est particulièrement important pour des instructions telles que INPUT et LINE INPUT dont le temps d'exécution dépend essentiellement de l'utilisateur.

- Il n'y a pas de sous-programme contenant l'instruction DI en cours d'exécution. En effet, l'instruction DI placée au début d'un sous-programme interdit son interruption par une instruction AFTER...GOSUB ou EVERY...GOSUB tant que la commande EI ou la fin du sous-programme (RETURN) n'est pas rencontrée.

Le temps d'exécution du sous-programme appelé par l'instruction EVERY...GOSUB ne doit pas être supérieur ou égal à l'intervalle de temps n1 ; dans le cas contraire, le sous-programme sera exécuté indéfiniment.

EXEMPLE DE PROGRAMME

Le programme associé à la commande AFTER...GOSUB illustre l'utilisation de l'instruction EVERY...GOSUB. L'utilisateur dispose de 10 secondes pour donner une réponse ; toutes les secondes, le temps restant est affiché à l'écran. La ligne 110 génère une interruption et l'exécution d'un sous-programme qui modifie la valeur associée au chronomètre.

ABS, ATN, CINT, COS, CREAL, EXP, FIX, INT, LOG, LOG10, MAX, MIN, ROUND, SGN, SIN, SQR, TAN, UNT

Les fonctions arithmétiques n'opèrent que sur des expressions algébriques (et non sur des chaînes de caractères). Elles fournissent un résultat numérique.

Toutes ces fonctions (à l'exception de MIN, MAX et ROUND) ont le même format : le mot clé désignant la fonction est immédiatement suivi de parenthèses contenant l'expression à calculer. Par exemple :

COS(X)   CINT(3.25)   EXP(3*(5+Y)/2)

Les fonctions arithmétiques peuvent être regroupées en différentes classes :

FONCTIONS TRIGONOMÉTRIQUES

Les fonctions trigonométriques sont les suivantes :

Fonction	Objet
ATN(X)	Calcule l'arc tangente de X
COS(X)	Calcule le cosinus de X
SIN(X)	Calcule le sinus de X
TAN(X)	Calcule la tangente de X

La fonction ATN(X) calcule l'arc tangente de X, c'est-à-dire l'angle dont la tangente est égale à X. Le résultat est exprimé par défaut en radians ; si la commande DEG a été exécutée auparavant, le résultat est obtenu en degrés. Il est aussi possible de multiplier le résultat par (180/3.141593), 180 degrés correspondant à 3.141593 radians.

L'argument X des autres fonctions trigonométriques est également exprimé par défaut en radians, l'instruction DEG permettant là aussi de changer le mode par défaut.

FONCTIONS DE TRONCATURE ET D'ARRONDI

Ces fonctions prennent comme argument X une expression numérique quelconque. Elles opèrent de la manière suivante :

Fonction	Objet
ABS(X)	Donne la valeur absolue de X
FIX(X)	Fournit la partie entière de X
INT(X)	Fournit le plus grand entier inférieur ou égal à X
ROUND(X,r)	Arrondit la partie décimale de X au nombre de chiffres après la virgule spécifié par r

Les fonctions FIX(X) et INT(X) sont très semblables, sauf lorsque l'argument X est un nombre négatif. Dans ce cas, la fonction FIX(X) renvoie l'entier supérieur (celui qui a la valeur absolue la plus faible), alors que la fonction INT(X) renvoie l'entier inférieur. Les limites autorisées pour les nombres entiers sont −32768 et 32767.

La fonction ROUND(X,r) n'a d'intérêt que pour les valeurs réelles de X (elle ne modifie pas un nombre entier). Les zéros non significatifs de la partie décimale sont supprimés par la fonction.

Exemples

```
PRINT ROUND(2.1234567,3)
```

Le résultat affiché est 2.123.

```
PRINT ROUND(-0.230001,4)
```

Le résultat est −0.23.

LES FONCTIONS DE CONVERSION DES NOMBRES (ENTIERS/RÉELS)

Deux fonctions permettent de convertir un nombre réel en nombre entier et réciproquement : CINT et CREAL. Une troisième fonction, UNT, permet de signer un nombre entier.

Fonction	Objet
CINT(X)	Convertit X en un nombre entier
CREAL(X)	Convertit X en un nombre réel
UNT(X)	Convertit un entier non signé X en un entier signé

Les nombres réels sont compris entre −1.7E−38 et 1.7E+38 (la lettre E symbolise la notation scientifique et se lit ''10 puissance''). Le plus petit nombre positif est 2.9E−39. Les nombres décimaux peuvent comporter jusqu'à dix chiffres après la virgule (en l'occurrence le point, en notation informatique).

Les nombres entiers algébriques ont toujours un signe (même si celui-ci n'est pas nécessairement affiché, le signe étant facultatif pour les nombres positifs). Il n'en est pas de même pour les nombres exprimant une adresse mémoire. Aussi est-il nécessaire de leur donner un signe (de les *signer*) pour pouvoir les manipuler dans certaines expressions algébriques. C'est précisément l'objet de la fonction UNT (qui ne peut opérer que sur des entiers décimaux, hexadécimaux ou binaires, en spécifiant les préfixes &H ou &X correspondants).

FONCTIONS TRANSCENDANTES

Ces fonctions retournent un résultat sous la forme d'un nombre réel.

Fonction	Objet
EXP(X)	Calcule l'exponentielle de X
LOG(X)	Calcule le logarithme naturel (ou népérien) de X
LOG10(X)	Calcule le logarithme décimal de X

L'argument X de la fonction EXP(X) doit être inférieur à 88,1 sous peine de débordement de capacité (Overflow).

L'argument X des fonctions LOG(X) et LOG10(X) doit nécessairement être un nombre positif.

AUTRES FONCTIONS ARITHMÉTIQUES

Les fonctions arithmétiques du BASIC Amstrad n'entrant pas dans les catégories précédemment établies sont les suivantes :

Fonction	Objet
MAX(X1,...,Xn)	Retourne la plus grande des valeurs parmi les arguments X1 à Xn fournis
MIN(X1,...,Xn)	Retourne la plus petite des valeurs parmi les arguments X1 à Xn fournis
SGN(X)	Permet de connaître le signe de X
SQR(X)	Calcule la racine carrée de X

Les arguments fournis aux fonctions MAX et MIN peuvent être des constantes ou des expressions algébriques (valeurs réelles ou entières).

La fonction SGN(X) renvoie la valeur 1 si X est positif, 0 si X est nul et –1 si X est négatif.

Toutes les fonctions mathématiques usuelles (arcsin, cosh, sinh, etc.) ne sont pas préprogrammées en BASIC Amstrad. Il est cependant possible de définir toute fonction supplémentaire de son choix, au moyen de l'instruction DEF FN (voir la description de cette instruction).

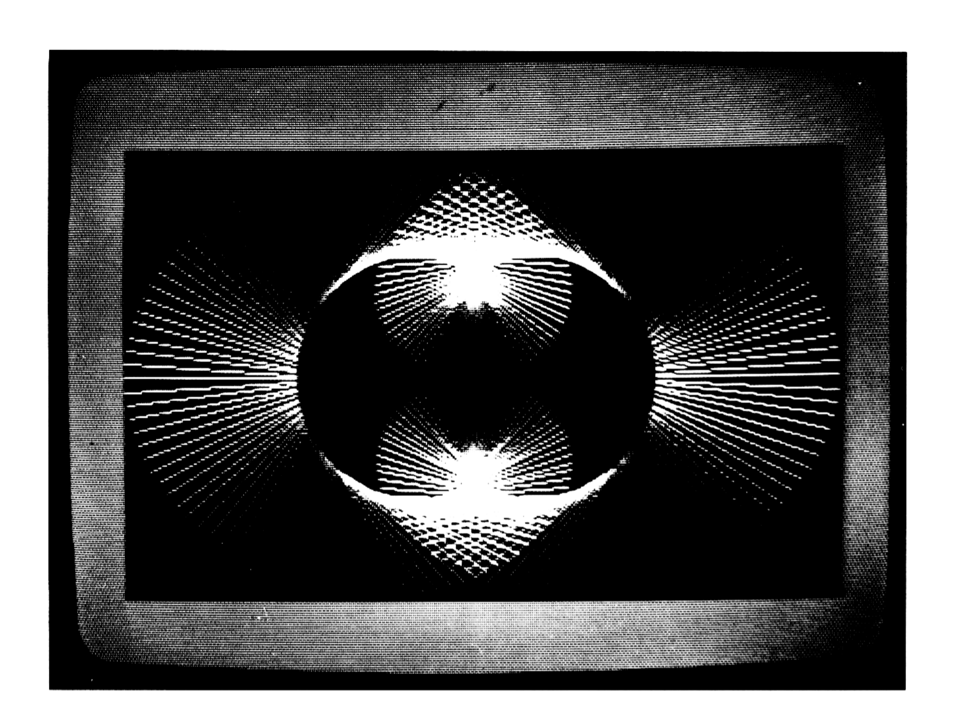
Le tableau suivant indique les formules permettant de calculer certaines de ces fonctions au moyen des fonctions arithmétiques intégrées au BASIC Amstrad :

Fonction	Équivalent Amstrad
Sécante	SEC(X) = 1/COS(X)
Cosécante	CSC(X) = 1/SIN(X)
Cotangente	COTG(X) = 1/TAN(X)
Arcsinus	ARCSIN(X) = ATN(X/SQR(–X*X–1))
Arccosinus	ARCCOS(X) = –ATN(X/SQR(–X*X+1)) + 1.5708
Arccotangente	ARCCOT(X) = ATN(X) + 1.5708
Sinus hyperbolique	SINH(X) = (EXP(X) – EXP(–X))/2
Cosinus hyperbolique	COSH(X) = (EXP(X) + EXP(–X))/2
Tangente hyperbolique	TANH(X) = EXP(–X)/(EXP(X) + EXP(–X))*2 + 1
Cotangente hyperbolique	COTH(X) = EXP(–X)/(EXP(X) – EXP(–X))*2 + 1
Arcsinus hyperbolique	ARCSINH(X) = LOG(X + SQR(X*X + 1))
Arccosinus hyperbolique	ARCCOSH(X) = LOG(X + SQR(X*X – 1))
Arctangente hyperbolique	ARCTANH(X) = LOG((1 + X)/(1 – X))/2
Arccotangente hyperbolique	ARCCOTH(X) = LOG((X + 1)/(X – 1))/2

EXEMPLE DE PROGRAMME

Le programme ci-dessous utilise (lignes 130 à 160) les fonctions SIN et COS pour définir les coordonnées de deux points de l'écran (P0 de coordonnées X0 et Y0 et P1 de coordonnées X1 et Y1). Le stylo est ensuite placé sur le premier point (ligne 170) puis une droite est tracée entre les deux points (ligne 180). Cette boucle d'instructions est réitérée pour des valeurs de I variant de 0 à 2 PI avec un pas d'incrémentation de PI/128 (lignes 120 à 190).

```
10 CLS : CLG
20 MODE 2
30 BORDER 9
40 GRAPHICS PAPER 0
50 INK 0,0
60 GRAPHICS PEN 1
70 INK 1,7
80 X=320
90 Y=200
100 H=320
110 V=51/64*H
120 FOR I=0 TO 2*PI STEP PI/128
130 X0=X+H/2*COS(I)
140 Y0=Y+V/2*SIN(I)
150 X1=X+H*COS(2*I)*COS(I)
160 Y1=Y+V*SIN(2*I)*COS(I)
170 MOVE X0,Y0
180 DRAW X1,Y1
190 NEXT I
200 C$=INKEY$ : IF C$="" THEN 200
210 MODE 1
220 INK 0,1
230 INK 1,24
240 END
```





Instruction
(CPC 664 et CPC 6128)

L'instruction FILL colore, à partir de la position courante du curseur graphique, une zone fermée de l'écran.

FORMAT

FILL numéro d'encre

- Le **numéro d'encre** est une valeur comprise entre 0 et 15 ; il caractérise l'instruction INK associée.

COMMENTAIRES

Si aucune zone n'a été définie, la totalité de l'écran est colorée, à l'exception des lignes qui ont été tracées à l'aide des instructions DRAW et DRAWR. De même, si le curseur graphique est placé à l'intérieur d'une figure qui n'est pas fermée, la couleur "déborde" et se répand à l'extérieur.

EXEMPLE DE PROGRAMME

Le programme ci-après illustre l'utilisation de la commande FILL. La ligne 60 colore la totalité de l'écran avec l'encre n° 3 qui a la couleur rouge vif, puis un cercle noir (qui n'est pas complètement fermé) est tracé au milieu de l'écran (lignes 110 à 130). Le curseur graphique est placé ensuite au centre du cercle (ligne 140) et l'intérieur de la figure est rempli avec une encre jaune (ligne 150).

On remarque que la zone colorée est limitée par les lignes tracées par les commandes DRAW précédentes ; lors de la première itération, la zone n'est pas fermée (le paramètre J varie entre 0 et 1,98 PI) et l'encre déborde à l'extérieur du cercle pour remplir la totalité de l'écran. Lors de la seconde itération, le paramètre J varie de 0 à 2 PI et le cercle est complètement fermé.

```
10 CLS
20 BORDER 6
30 INK 3,6
40 INK 2,0
50 FOR J=1 TO 2
60 FILL 3
70 GRAPHICS PEN 2
80 ORIGIN 320,200
90 MOVE 140,0
100 IF J=2 THEN F=2*PI ELSE F=2*PI-(2*PI
/100)
110 FOR I= 0 TO F STEP 2*PI/100
120 DRAW COS(I)*140,SIN(I)*140
130 NEXT I
140 MOVE 0,0
150 FILL 1
160 LOCATE 1,25
170 PRINT "Tapez une touche";
180 C$=""
190 WHILE C$="" : C$=INKEY$ : WEND
200 CLS
210 NEXT J
```

Les instructions FOR et NEXT permettent d'exécuter, de manière répétitive, une suite d'instructions pour différentes valeurs d'une variable incrémentée par l'argument de STEP.

FORMATS

FOR variable=x TO y < STEP z >

.

.

.

NEXT < variable > < ,variable > ...

- **variable** est une variable numérique utilisée comme compteur.
- **x** est une expression numérique qui positionne la valeur initiale du compteur.
- **y** est une expression numérique qui fixe la valeur finale du compteur.
- **z** est une expression numérique qui spécifie l'incrément de la variable.

RÈGLES DE SYNTAXE ET COMMENTAIRES

STEP z est optionnel ; s'il est omis, la valeur par défaut de l'incrément est égale à 1.

Les instructions figurant entre FOR (TO) et NEXT sont exécutées en prenant x comme valeur de la variable. Le compteur est ensuite incrémenté et la suite d'instructions est à nouveau exécutée, avec cette fois (x+z) comme valeur de la variable. L'opération se poursuit tant que la valeur de la variable n'excède pas y. Quand cela se produit, les instructions qui suivent l'instruction NEXT sont à leur tour exécutées (sortie de la boucle FOR/NEXT).

z peut être négatif, auquel cas la variable est décrémentée de x à y. Des valeurs de x, y et z incompatibles ne provoquent ni mes-

sage d'erreur ni interruption de l'exécution du programme ; la boucle FOR/NEXT et les instructions qui y figurent sont simplement ignorées. Il y a incompatibilité dans les deux cas suivants :

- lorsque x est supérieur à y, z étant positif ;
- lorsque x est inférieur à y, z étant négatif.

Il est recommandé, dans la mesure du possible, d'initialiser le compteur avec des nombres entiers (bien que les valeurs décimales soient également admises). La spécification du nom de la variable après l'instruction NEXT est optionnelle (elle ralentit d'ailleurs légèrement la vitesse d'exécution du programme). Cette spécification est cependant nécessaire en cas d'utilisation de plusieurs boucles FOR/NEXT imbriquées (c'est-à-dire lorsque des boucles FOR/NEXT sont insérées à l'intérieur d'autres boucles FOR/NEXT). Des boucles imbriquées doivent avoir pour compteurs des variables de noms différents. L'instruction NEXT correspondant à la boucle intérieure doit toujours apparaître avant celle qui correspond à la boucle extérieure. Cependant, dans le cas où des boucles imbriquées ont le même point final, une seule instruction NEXT peut être utilisée pour toutes les variables. Par exemple, les lignes consécutives :

```
110 NEXT I
120 NEXT J
130 NEXT K
```

peuvent être remplacées par la ligne :

```
110 NEXT I,J,K
```

Dans un programme, l'instruction FOR doit toujours être rencontrée avant l'instruction NEXT correspondante. Dans le cas contraire, le message d'erreur "Unexpected NEXT" (NEXT mal positionné) est affiché.

EXEMPLE DE PROGRAMME

Cet exemple illustre une manière possible de remplir et d'afficher à l'écran un tableau à deux dimensions. Chaque élément de ce tableau, B$(I,J), représente le résultat (sous forme d'une chaîne de caractères) d'une rencontre sportive ayant opposé les deux équipes

nationales I et J. Quatre équipes A$(I) ont participé à la compétition, chacune ayant rencontré les trois autres deux fois (matchs aller et retour).

Les résultats des rencontres sont tout d'abord entrés au clavier, en réponse aux questions affichées. Dès que les résultats des douze rencontres sont entrés, le programme les affiche sous forme d'un tableau synoptique. Pour retrouver le résultat d'une rencontre particulière, par exemple Italie-Espagne, il suffit de regarder la valeur indiquée à l'intersection de la ligne Italie et de la colonne Espagne. Le format d'impression donné dans le programme est optimal pour des résultats exprimés sous la forme x – y (1 – 0, 5 – 3, etc.). La seconde partie du programme (lignes 240 à 420) permet d'établir le classement des quatre équipes en fonction des résultats obtenus. Ce classement n'est exact que dans le cas où le format x – y a été respecté pour l'entrée des résultats. Cette seconde partie de programme est explicitée en illustration des fonctions LEFT$ et RIGHT$.

```
10 CLS
20 A$(1)="ESPAGNE" : A$(2)="FRANCE " : A
$(3)="ITALIE " : A$(4)="SUISSE "
30 PRINT "ENTREZ LE RESULTAT DES RENCONT
RES" : PRINT : PRINT
40 FOR I=1 TO 4
50 FOR J=1 TO 4
60 IF I=J THEN B$(I,J)=" -" ELSE PRINT
   A$(I);"-";A$(J); : INPUT B$(I,J)
70 B$(I,J)=" "+B$(I,J)
80 NEXT J
90 NEXT I
100 '
110 '************************************
120 '    MISE EN FORME DU TABLEAU
130 '************************************
140 CLS
150 PRINT:PRINT TAB(9) A$(1);TAB(17) A$(
2);TAB(25) A$(3);TAB(33) A$(4)
160 FOR I=1 TO 4
165 PRINT : PRINT
170 FOR J=0 TO 4
180 FOR K=1 TO 4 : B$(K,0)=A$(K) : NEXT
K
190 IF POS(#0)>35 THEN PRINT CHR$(13)
200 PRINT TAB(8*J) B$(I,J);
210 NEXT J
220 NEXT I
230 '
```

```
240 '********************************
250 '   ETABLISSEMENT DU CLASSEMENT
260 '********************************
270 FOR I=1 TO 4
280 FOR J=1 TO 4
290 B$(I,J)=RIGHT$(B$(I,J),3)
300 IF I<>J THEN IF LEFT$(B$(I,J),1) > R
IGHT$(B$(I,J),1) THEN A(I)=A(I)+2 ELSE I
F LEFT$(B$(I,J),1)<RIGHT$(B$(I,J),1) THE
N A(J)=A(J)+2 ELSE A(I)=A(I)+1:A(J)=A(J)
+1
310 NEXT J
320 NEXT I
330 PRINT : PRINT
340 FOR I=1 TO 3
350 FOR J=(I+1) TO 4
360 IF A(I)<A(J) THEN D=A(I) : A(I)=A(J)
 : A(J)=D : D$=A$(I) : A$(I)=A$(J) : A$(
J)=D$
370 NEXT J
380 NEXT I
390 INK 1,26:INK 0,1
400 PRINT : PRINT : PRINT "Classement" :
 PRINT : PRINT
410 FOR I=1 TO 4 : PRINT A$(I),A(I) : NE
XT I
420 END
```

Lignes 40 à 90

Les résultats sont entrés au clavier et stockés dans les variables B$(I,J) au moyen de deux boucles FOR/NEXT imbriquées. L'instruction IF...THEN...ELSE de la ligne 60 permet d'éviter la demande du résultat d'une rencontre qui ne peut s'être déroulée (c'est-à-dire lorsque les variables I et J correspondent à la même équipe). Dans ce cas, un tiret (précédé d'un espace afin de conserver un format d'impression correctement centré) est stocké dans la variable B$(I,J) correspondante. La ligne 70 ajoute deux espaces à gauche de chaque résultat (afin que ceux-ci soient mieux centrés dans le tableau, en regard du format choisi pour l'affichage du nom des équipes).

Ligne 150

Affichage sur une ligne du nom des quatre équipes, en utilisant la fonction TAB.

Lignes 160 à 220

Affichage des résultats, chaque ligne commençant par le nom d'une des quatre équipes. Trois boucles FOR/NEXT imbriquées sont utilisées à cet effet. La boucle centrale (ligne 180) permet d'affecter le nom contenu dans la variable A$(K) à la variable correspondante B$(K,0). Cela ne modifie rien aux résultats déjà stockés, puisque les variables B$(I,J) d'indice 0 n'ont pas été utilisées à ce propos. De cette manière, chaque ligne du tableau (écrite par la boucle FOR/NEXT ayant J pour variable) commence effectivement par un nom d'équipe (J=0) et continue par quatre résultats, le format de l'affichage étant défini par l'instruction PRINT TAB de la ligne 200. La ligne 190 permet d'ajouter un retour chariot, dont le code ASCII est CHR$(13), après l'affichage du quatrième résultat, ce qui correspond à un changement de ligne pour chaque valeur différente de la variable I.

Instruction
(CPC 664 et CPC 6128)

FRAME

L'instruction FRAME est destinée à synchroniser l'affichage de graphismes sur l'écran.

FORMAT

FRAME

COMMENTAIRES

L'instruction FRAME produit un effet de *lissage du mouvement* lors de l'affichage de caractères en mode graphique (associés à la commande TAG) ; leur apparition sur l'écran est moins saccadée.

EXEMPLE DE PROGRAMME

Le programme illustrant les commandes TAG et TAG OFF emploie (ligne 390) l'instruction FRAME. Celle-ci est réitérée avant chaque commande MOVE X2,1,2,1 qui déplace le curseur graphique ; elle est destinée à donner un aspect plus naturel et continu au déplacement de la voiture.





La fonction FRE indique le nombre d'octets en mémoire centrale non occupés par les programmes et les données écrits en BASIC.

FORMATS

FRE(a)

FRE(a$)

- **a** et **a$** sont des arguments fictifs, expression numérique (a) ou chaîne de caractères (a$).

COMMENTAIRES

La valeur renvoyée par la fonction FRE(a) ne dépend pas de l'expression numérique a. De même, quelle que soit la chaîne de caractères a$, FRE(a$) retourne toujours la même valeur. Une chaîne x$ peut comporter jusqu'à 255 caractères.

Un exemple d'utilisation de cette fonction est donné en illustration de l'instruction ERASE. Le programme d'application permet de calculer la place mémoire occupée par des tableaux numériques de différentes tailles. On remarque qu'une variable numérique est stockée sur cinq octets et qu'une variable indicée (tableau) nécessite douze octets pour le descriptif du tableau.

Lorsque la fonction FRE prend la forme suivante :

FRE ("")

le résultat n'est pas affiché immédiatement ; le système effectue au préalable une récupération de l'espace non utilisé. En effet, étant donné que les variables alphanumériques peuvent avoir des tailles variant entre 1 et 255 octets, le BASIC gère de manière dynamique l'espace réservé aux variables alphanumériques et, chaque fois qu'une nouvelle valeur est assignée à la variable, l'adresse de sa zone de stockage varie. Ces adresses augmentent jusqu'à la limite de l'espace

disponible pour stocker les variables ; le BASIC effectue ensuite automatiquement une récupération de l'espace inutilisé afin de *compacter* la zone réservée au stockage des variables. La fonction FRE("") force cette réorganisation.

ERR, ERL, ERROR, ON ERROR GOTO, RESUME

Ce groupe de fonctions et d'instructions permet de gérer les erreurs susceptibles de se produire lors de l'exécution d'un programme.

Les fonctions ERR et ERL fournissent respectivement les numéros de code et de ligne associés à une erreur.

L'instruction ERROR permet à l'utilisateur soit de définir ses propres codes d'erreur, soit de simuler l'apparition d'une erreur BASIC.

L'instruction ON ERROR GOTO dirige l'exécution du programme, lorsqu'une erreur est rencontrée, vers le sous-programme de gestion des erreurs.

L'instruction RESUME permet de reprendre l'exécution du programme principal après que la procédure de correction d'erreur a été mise en œuvre.

FORMATS

Le format de chacune de ces variables ou instructions est le suivant :

ERR

ERL

ERROR expression entière

- L'**expression entière** est comprise entre 0 et 255.

ON ERROR GOTO numéro de ligne

RESUME < 0 >

ou RESUME NEXT

ou RESUME numéro de ligne

Variables ERR et ERL

ERR et ERL sont des variables dont le nom est réservé (et non pas des instructions). Elles ne peuvent donc être utilisées à d'autres fins que celles qui sont définies ci-dessous ; il est par exemple impossible de leur assigner une valeur au moyen d'une instruction d'assignation.

Lorsqu'une erreur est décelée, l'exécution du programme est redirigée vers une section BASIC chargée de gérer les conditions d'erreur. A ce moment, la variable ERR reçoit comme valeur le numéro de code correspondant à l'erreur décelée et la variable ERL le numéro de la ligne où l'erreur s'est produite. Ces deux variables sont habituellement utilisées dans des instructions IF...THEN pour transférer le contrôle au sous-programme chargé de gérer les erreurs.

Pour tester une condition d'erreur dans un programme, il est conseillé de respecter la syntaxe suivante :

IF ERL = numéro de ligne THEN...

c'est-à-dire de placer le numéro de ligne à droite du signe égal. La syntaxe inverse (numéro de ligne = ERL) n'est pas incorrecte. Elle a cependant l'inconvénient suivant : si les lignes de programme sont renumérotées au moyen de la commande RENUM, le numéro de ligne donné comme valeur à ERL ne sera également modifié que si la première syntaxe a été employée.

Lorsqu'une erreur se produit en mode direct, la variable ERL contient la valeur 65535. Pour tester l'éventualité d'une telle erreur, il convient d'utiliser l'instruction particulière :

IF 65535 = ERL THEN...

Instruction ERROR

Si la valeur donnée comme argument à cette instruction est égale à l'un des codes d'erreur utilisé par le BASIC, ERROR simule l'apparition de cette erreur. Si un sous-programme de gestion d'erreur a été défini par une instruction ON ERROR GOTO, ce sous-programme est exécuté. Dans le cas contraire, le message correspondant au code d'erreur est affiché et l'exécution s'arrête.

Les codes des messages d'erreur sont donnés en annexe du guide de l'utilisateur livré avec l'ordinateur. Par exemple, les codes d'erreur BASIC de l'Amstrad CPC 464 sont compris entre 1 et 30. L'utilisateur pourra ainsi redéfinir tout message de son choix dont le numéro de code sera compris entre 31 et 255. Il convient toutefois de prendre garde, sur les modèles CPC 664 et CPC 6128, qu'un certain nombre de codes supplémentaires (les codes 30 et 31 en particulier) sont réservés aux erreurs disque (voir la fonction DERR).

Pour définir son propre code d'erreur, il suffit de donner comme argument à l'instruction ERROR un numéro de code qui n'est pas déjà utilisé par le BASIC (ou par AMSDOS ou par le contrôleur de disquette, sur les modèles CPC 664 et CPC 6128). Ce nouveau code peut alors être testé, comme les autres codes, par un sous-programme de gestion des erreurs (voir l'exemple donné ci-après).

Instruction ON ERROR GOTO

Dès que cette instruction est rencontrée en cours d'exécution du programme, toute erreur (même une erreur en mode direct) est dirigée vers le sous-programme de gestion des erreurs, dont la première ligne doit constituer l'argument de cette instruction. Si l'argument correspond à un numéro de ligne qui n'existe pas, le message "Line does not exist" (numéro de ligne non défini) est affiché.

L'instruction particulière ON ERROR GOTO 0 a l'effet inverse : elle empêche le déroutement du programme en cas d'erreur. Les erreurs éventuelles rencontrées après l'exécution de cette instruction provoquent simplement l'affichage d'un message d'erreur et l'arrêt de l'exécution du programme. Il peut être utile d'inclure une telle instruction dans le sous-programme de gestion des erreurs, pour toute erreur pour laquelle aucune procédure de correction n'est prévue.

Instruction RESUME

RESUME est une instruction spécifiquement implantée dans un sous-programme de gestion des erreurs. Rencontrée ailleurs dans le programme, elle provoque l'affichage du message "Unexpected RESUME" (RESUME sans gestion d'erreur). Selon le format utilisé, cette instruction a les effets suivants :

RESUME ou **RESUME 0**

L'exécution du programme principal reprend à partir de l'instruction qui a provoqué l'erreur.

Il est conseillé d'utiliser le format RESUME de préférence à la forme équivalente RESUME 0 car, dans le second cas, un message d'erreur ''Line does not exist'' (ligne non définie) risque d'être affiché en cas de renumérotation des lignes du programme au moyen de la commande RENUM (la ligne 0 n'est en effet généralement pas utilisée).

```
RESUME NEXT
```

L'exécution du programme principal reprend à l'instruction qui suit celle qui a causé l'erreur.

```
RESUME numéro de ligne
```

L'exécution reprend au numéro de ligne spécifié.

EXEMPLE DE GESTION D'ERREUR

Cet utilitaire de gestion d'erreur est destiné au programme de jeu donné en illustration de la rubrique ''Opérateurs logiques''. Les quatre lignes ci-dessous doivent être ajoutées à celles du programme de jeu, sans rien modifier à celui-ci par ailleurs.

Il s'agit de traiter les erreurs qui se produiraient si un joueur entrait une valeur et/ou une couleur de carte ne correspondant pas à celle(s) d'un jeu classique de 52 cartes.

```
85 ON ERROR GOTO 300
.
.
.
92 IF NOT (Y(N) < 14 AND Y(N) > 0) THEN ERROR 220
.
.
.
115 IF NOT (X(N) > 0 AND X(N) < 5) THEN ERROR 220
.
.
.
300 IF ERR=220 AND ERL=92 THEN PRINT "La valeur proposée n'existe pas.
```

```
Recommencez":RESUME 90 ELSE PRINT "La couleur proposée n'existe pas.
Recommencez":RESUME 100
```

La ligne 85 indique que le sous-programme de gestion d'erreur se trouve à la ligne 300. Toute erreur décelée après que la ligne 85 a été exécutée redirigera le programme vers la ligne 300.

Si la valeur de la variable Y(N) n'est pas comprise entre 1 et 13, une erreur ayant 220 pour numéro de code sera détectée à la ligne 92. De même, si la variable X(N) n'est pas comprise entre 1 et 4, une erreur (code 220) sera décelée à la ligne 115. D'après ce qui précède, l'exécution sera, dans l'un et l'autre cas, redirigée vers la ligne 300.

La clause THEN de la ligne 300 est exécutée si les deux conditions ERR = 220 et ERL = 92 sont simultanément réalisées, c'est-à-dire si la valeur de la carte entrée à la ligne 92 est erronée (d'où le message imprimé par la clause THEN). Dans le cas contraire (qui, d'après la ligne 115, correspond à ERR = 220 et ERL = 115), la clause ELSE sera exécutée (erreur sur la couleur entrée).

L'instruction GOSUB détourne l'exécution d'un programme vers celle d'un sous-programme. L'instruction RETURN renvoie l'exécution vers le programme principal.

FORMATS

GOSUB numéro de ligne
.
.
.
RETURN

- Le **numéro de ligne** est celui de la première ligne du sous-programme.

COMMENTAIRES

Lorsqu'une instruction GOSUB est rencontrée au cours de l'exécution d'un programme, celui-ci est immédiatement dirigé vers le sous-programme dont le numéro de la première ligne est spécifié dans l'instruction GOSUB. Le sous-programme est exécuté jusqu'à ce qu'une instruction RETURN soit rencontrée. A ce moment, l'exécution du programme principal reprend à la première instruction se trouvant après GOSUB.

Un sous-programme se distingue donc du programme principal par la présence, comme dernière instruction, de RETURN. Par contre, aucune instruction particulière n'indique qu'une ligne quelconque constitue la première ligne d'un sous-programme. Si le sous-programme est inséré à n'importe quel endroit du programme principal, il risque d'être exécuté inopportunément lorsque la ligne du programme principal qui le précède aura elle-même été exécutée. Cela peut être évité si l'on écrit, entre le sous-programme et le programme principal, une instruction permettant de contourner le sous-programme (par exemple GOTO, STOP ou END, selon le cas). La lisibilité d'un listing est cependant améliorée lorsque les sous-

programmes sont placés après le programme principal, avec un décalage important au niveau des numéros de ligne.

Un sous-programme peut être appelé aussi souvent que nécessaire par le programme principal. Il peut d'autre part comporter plusieurs instructions RETURN, certaines s'exécutant de manière conditionnelle, au moyen d'une instruction IF...THEN par exemple.

Un sous-programme peut en appeler un autre, l'instruction RETURN renvoyant toujours au programme ou au sous-programme appelant. L'instruction ON...GOSUB permet le branchement conditionnel vers différents sous-programmes, en fonction de la valeur de l'expression spécifiée par ON.

Le programme donné en illustration de l'instruction ERASE constitue un exemple d'utilisation d'un sous-programme.

L'instruction GOTO déclenche un branchement inconditionnel vers une ligne spécifiée ; l'exécution normale du programme reprend (en mode direct) ou continue (en mode programme) à partir de la ligne spécifiée.

FORMAT

GOTO numéro de ligne

COMMENTAIRES

Le numéro de la ligne spécifiée peut éventuellement contenir des instructions non exécutables, telles que REM ou DIM. Dans ce cas, le programme se poursuit jusqu'à trouver une instruction exécutable. Le mot clé GOTO peut également être utilisé dans des instructions provoquant des branchements conditionnels (ON...GOTO, IF...GOTO), c'est-à-dire que le programme n'est dirigé vers la ligne spécifiée après GOTO que si une condition est réalisée (IF...GOTO). Il peut également être dirigé vers différentes lignes selon la valeur d'une expression (ON...GOTO). Lorsqu'il est utilisé seul, comme branchement inconditionnel, le mot clé GOTO constitue à lui seul une instruction. Par contre, dans un branchement conditionnel, l'instruction est ON...GOTO ou IF...GOTO (voir la description de ces instructions pour de plus amples informations).

Il est conseillé, dans la mesure du possible, d'éviter l'emploi répété de l'instruction GOTO dans un programme et de la réserver plutôt à une utilisation en mode direct (pour la mise au point par exemple). En effet, le listing d'un programme comportant un grand nombre d'instructions GOTO peut s'avérer difficile et long à déchiffrer (même si l'on a écrit soi-même le programme), dans la mesure où les opérations réalisées successivement par le programme sont dispersées et enchevêtrées au milieu de lignes de programme intervenant à une autre étape de l'exécution. Il est préférable de structurer l'organisation d'un programme afin que ce qu'il réalise puisse être compris (et éventuellement modifié) à la simple lecture de son listing.

GRAPHICS PAPER et GRAPHICS PEN

Instructions (CPC 664 et CPC 6128)

Les instructions GRAPHICS PAPER et GRAPHICS PEN initialisent respectivement la couleur de l'encre de fond et la couleur du stylo pour le mode graphique.

FORMATS

GRAPHICS PAPER numéro d'encre

GRAPHICS PEN numéro d'encre < ,mode du fond >

- Le **numéro d'encre** est un nombre entier compris entre 0 et 15.
- Le **mode du fond** prend la valeur 0 ou 1. 0 correspond à un fond opaque, tandis que 1 correspond à un fond transparent.

COMMENTAIRES

La mise en œuvre des instructions GRAPHICS PAPER et GRAPHICS PEN est identique à celle des instructions PAPER et PEN.

Lorsque l'instruction GRAPHICS PEN est utilisée avec l'instruction TAG, les caractères écrits à un endroit quelconque de l'écran peuvent avoir un fond visible ou transparent (mode du fond 0 ou 1).

EXEMPLE

Le programme ci-après dessine une carte de France rouge sur un fond bleu. Les lignes 10 à 90 initialisent l'écran ; la couleur du fond est bleue et le stylo a la couleur noire. Ensuite, la carte est tracée et l'intérieur du dessin est rempli avec la couleur rouge (lignes 570 et 580). La figure est fermée afin que, lors du remplissage, la couleur ne déborde pas.

Les lignes 1020 à 1060 écrivent l'expression "Carte de France" dans le bas de l'écran ; on remarque (ligne 1030) que le mode du fond

est opaque, ce qui permet d'écrire les lettres sur un fond rouge (ligne 1020). La ligne 1030 peut être modifiée de la manière suivante :

```
1030 GRAPHICS PEN 1,1
```

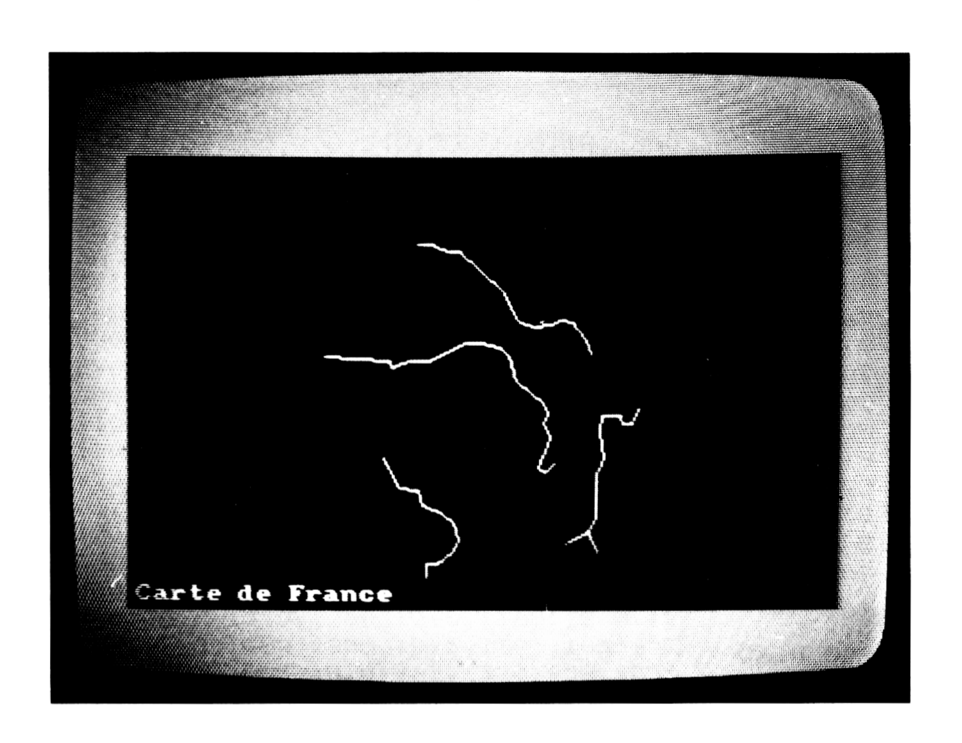
Le mode du fond est alors transparent ; les caractères sont écrits sur fond bleu.

```
10 CLS
20 MODE 1
30 INK 0,1
40 GRAPHICS PAPER 0
50 INK 2,0
60 GRAPHICS PEN 2
70 BORDER 7
80 ORIGIN 70,-30
90 CLG
100 '
110 ' PAS DE CALAIS - COTENTIN
120 '
130 DATA 850,1430,755,1400,750,1300,705,
1262,652,1250,615,1225,620,1200,610,1180
,575,1165,490,1180,470,1200,470,1230,435
,1225,405
140 '
150 ' COTENTIN - FINISTERE
160 '
170 DATA 1238,410,1200,430,1170,430,1110
,445,1065,395,1088,370,1070,358,1086,315
,1065,275,1115,232,1110,228,1092,145,109
5
180 '
190 ' FINISTERE - BRETAGNE SUD
200 '
210 DATA 109,1080,105,1050,145,1058,140,
1040,130,1040,126,1048,120,1035,130,1020
,135,1030,150
220 '
230 ' BRETAGNE SUD - VENDEE
240 '
250 DATA 1020,100,1008,140,988,145,960,1
75,975,195,960,260,925,322,875,360,870,3
45,835
260 '
270 ' VENDEE - PYRENNEES
280 '
290 DATA 370,820,440,728,460,635,520,560
,510,560,450,610,440,490,460,490,455,475
```

→

```
,440,475,400,320,368,285,405,270,400,240
300 '
310 ' PYRENNEES - MEDITERRANEE
320 '
330 DATA 500,200,540,200,560,180,640,180
,635,200,790,125,820,190,850,120,920,140
340 '
350 ' MEDITERRANEE - COTE D'AZUR
360 '
370 DATA 910,240,1015,309,1080,280,1170,
260,1220,240,1320,265,1385,343,1420,405
380 '
390 ' ITALIE - ALLEMAGNE
400 '
410 DATA 1330,440,1335,500,1295,550,1342
,590,1300,680,1300,740,1230,705,1225,754
,1310,880,1310,905,1370,910,1420,1130,14
05,1175
420 '
430 ' LUXEMBOURG - PAS DE CALAIS
440 '
450 DATA 1330,1160,1260,1160,1250,1200,1
150,1200,1080,1255,1075,1305,1055,1270,1
010,1270,1005,1320,962,1320,930,1350,905
,1390,875,1380,850,1430
460 '
470 EX=640/2000
480 EY=400/1350
490 READ X,Y
500 MOVE X*EX,Y*EY
510 FOR I=2 TO 103
520 READ X,Y
530 DRAW X*EX,Y*EY,2
540 NEXT I
550 MOVE 320,200
560 '
570 INK 3,3
580 FILL 3
590 '
600 GRAPHICS PEN 1
610 '
620 ' SEINE
630 '
640 DATA 620,1200,660,1200,710,1180,740,
1180,860,1065,900,980,925,965,950,960,99
0,970,1020,980,1050,975,1090,920,1100,88
0
650 READ X,Y
660 MOVE X*EX,Y*EY
670 FOR I=1 TO 12
680 READ X,Y
```

```
690 DRAW X*EX,Y*EY,1
700 NEXT I
710 '
720 ' LOIRE
730 '
740 DATA 360,870,545,860,550,840,600,860
,650,860,760,910,800,915,850,900,875,870
,895,800,970,730,980,700,970,670,990,625
,960,530,975,515,990,525,1000,540
750 READ X,Y
760 MOVE X*EX,Y*EY
770 FOR I=1 TO 17
780 READ X,Y
790 DRAW X*EX,Y*EY
800 NEXT I
810 '
820 ' RHONE
830 '
840 DATA 1230,710,1215,665,1190,660,1180
,690,1135,690,1125,640,1140,565,1120,510
,1120,380,1100,335,1040,300,1100,335,112
5,270
850 READ X,Y
860 MOVE X*EX,Y*EY
870 FOR I=1 TO 12
880 READ X,Y
890 DRAW X*EX,Y*EY
900 NEXT I
910 '
920 ' GARONNE
930 '
940 DATA 525,560,570,470,620,460,635,415
,685,390,720,360,730,315,715,270,670,240
,640,240,640,200
950 READ X,Y
960 MOVE X*EX,Y*EY
970 FOR I=1 TO 10
980 READ X,Y
990 DRAW X*EX,Y*EY
1000 NEXT I
1010 '
1020 GRAPHICS PAPER 3
1030 GRAPHICS PEN 1,0
1040 TAG
1050 MOVE -65,50
1060 PRINT "Carte de France";
1070 TAGOFF
1080 PAPER 0
1090 PEN 1
1100 C$=INKEY$ : IF C$="" THEN 1100
```


Carte de France

La fonction HIMEM retourne l'adresse de l'octet le plus haut de la mémoire disponible sous BASIC.

FORMAT

HIMEM

COMMENTAIRES

A l'initialisation du système, HIMEM renvoie la valeur 43903 (CPC 464) ou 42619 (CPC 664). La partie basse de la mémoire (adresse 0000H à 0040H) étant utilisée par le système, il reste néanmoins près de 42 K de mémoire RAM disponible pour l'utilisateur. 16 K supplémentaires sont réservés à la mémoire écran (mémoire haute, adresses C000H à FFFFH). L'instruction MEMORY permet d'abaisser la limite de la zone disponible pour le BASIC. Cela est particulièrement utile si l'on veut développer des programmes écrits en langage machine.

Il faut également savoir que l'ordinateur abaisse de 4 K la zone mémoire utilisable sous BASIC lorsqu'un fichier est ouvert sur cassette ou sur disquette. On peut s'en convaincre facilement en exécutant le petit programme suivant (après avoir placé une disquette ou une cassette dans le lecteur correspondant) :

```
10 H=HIMEM
20 OPENOUT "essai"
30 PRINT (H-HIMEM)/1024
```

La valeur affichée est 4, c'est-à-dire quatre kilo-octets. Cette place mémoire est utilisée comme tampon pour le fichier ouvert.

IF...THEN...ELSE
IF...GOTO...ELSE

Ces instructions constituent des tests conditionnels permettant, selon le résultat, de prendre une décision quant au déroulement du programme.

FORMATS

IF expression < , > THEN clause < ELSE clause >

IF expression < , > GOTO ligne < < , > ELSE clause >

Exemples

IF A > B THEN C=B ELSE P=Q

IF A$(1)=A$(2) THEN 110

IF A > B THEN IF B > C THEN PRINT "A > C" ELSE PRINT "B < =C" ELSE PRINT "A < =B"

RÈGLES DE SYNTAXE

L'expression est une expression numérique quelconque. Lorsqu'elle est vérifiée, la clause THEN ou GOTO est exécutée. La clause THEN peut contenir une instruction, une séquence de plusieurs instructions (séparées par :) ou un numéro de ligne. GOTO est nécessairement suivi d'un numéro de ligne.

Si l'expression n'est pas vérifiée, la clause THEN (ou GOTO) n'est pas prise en considération et la clause ELSE, si elle existe, est exécutée. Cette clause est facultative. Si elle est omise, l'exécution se poursuit normalement à partir de l'instruction suivante.

COMMENTAIRES

Les trois mots IF, THEN et ELSE (ou IF, GOTO et ELSE) ne forment qu'une seule instruction. Cela signifie que les clauses THEN et ELSE

doivent nécessairement figurer sur la même ligne de programme (définie par un numéro) que le mot clé IF auquel elles se rapportent.

Les instructions IF...THEN...ELSE peuvent être imbriquées, dans la limite maximale d'une ligne de programme. La clause ELSE étant facultative, il peut arriver qu'une ligne d'instructions contienne moins de mots ELSE que de mots THEN. Dans ce cas, chaque ELSE se rapporte au THEN non apparié le plus proche, en allant vers le début de la ligne. Le programme suivant illustre cette règle d'appariement.

EXEMPLE DE PROGRAMME

La partie de programme détaillée ici est incluse dans l'algorithme de tri donné en illustration de l'instruction MID$.

M mots ou expressions ont été entrés au clavier et stockés dans les variables de chaînes A$(I). En vue d'établir un classement de ces chaînes par ordre alphabétique, on souhaite éliminer dans chacune d'elles les blancs, les apostrophes et les tirets. C'est l'objet des instructions figurant de la ligne 140 à la ligne 200, telles qu'elles sont explicitées ci-dessous.

Ligne 170

La valeur de chaque variable A$(I) est passée à la variable C$(I) correspondante et la valeur " " (caractère blanc) est donnée à la variable B$.

Ligne 180

La première localisation du caractère blanc est recherchée dans la chaîne C$(I). Le numéro correspondant est stocké dans la variable numérique N. Si la chaîne C$(I) ne contient pas de blanc, N sera égal à 0.

Ligne 190

Cette ligne est réécrite ci-dessous en affectant un numéro entre parenthèses à chaque mot clé IF, THEN et ELSE. Ces numéros sont uniquement destinés à décrire la procédure d'appariement ; ils ne doivent en aucun cas être écrits dans le programme.

En fonction de la règle d'appariement précédemment énoncée :

IF(1) est apparié à THEN(1) et ELSE(3)
IF(2) est apparié à THEN(2) et ELSE(1)
IF(3) est apparié à THEN(3) et ELSE(2)

On notera d'autre part que les deuxième et troisième clauses THEN contiennent chacune deux instructions (séparées par :).

Supposons que la variable C$(I) considérée ait pour valeur l'expression "Programme essai". Un blanc se trouvant en dixième position dans cette expression, l'instruction de la ligne 180 fournira la valeur 10 à N. L'expression IF(1) n'est donc pas vérifiée, ce qui entraîne l'exécution de la clause ELSE(3), c'est-à-dire que la chaîne C$(I) est réécrite en supprimant le blanc ; elle devient "Programmeessai".

Lorsque la variable C$(I) testée ne contient ni blanc, ni tiret, ni apostrophe, l'ordre d'exécution est le suivant :

IF(1) vrai - THEN(1) - IF(2) vrai - THEN(2) - IF(1) vrai - THEN(1) - IF(2) faux - ELSE(1) - IF(3) vrai - THEN(3) - IF(1) vrai - THEN(1) - IF(2) faux - ELSE(1) - IF(3) faux - ELSE(2)

Ligne 200

L'expression N > 0 n'est vérifiée que si l'un des trois caractères — blanc, apostrophe ou tiret — a été rencontré (et supprimé) dans la chaîne C$(I). Dans ce cas, la clause THEN renvoie l'exécution du programme à la ligne 180, afin de rechercher une nouvelle occurrence du caractère déjà trouvé et celle des caractères non encore testés. Lorsque tous les caractères en question ont été éliminés (ou lorsque la chaîne n'en contenait pas à l'origine), N devient égal à 0, de sorte que la clause ELSE est exécutée, c'est-à-dire que l'ensemble des opérations est de nouveau effectué sur la chaîne C$(I) suivante (NEXT I).

On trouvera, dans le programme illustrant l'instruction MID$, un exemple d'utilisation de l'opérateur logique AND dans une boucle IF...THEN.



L'instruction INK permet d'attribuer une ou deux couleurs à un numéro d'encre donné.

FORMAT

INK numéro d'encre, couleur1 < ,couleur2 >

- Le **numéro d'encre** est une valeur comprise entre 0 et 15.
- **couleur1** est une valeur comprise entre 0 et 26 ; elle correspond à une des couleurs proposées dans la palette Amstrad.
- **couleur2** est une valeur comprise entre 0 et 26 ; elle correspond à la seconde couleur affichée par intermittence.

COMMENTAIRES

En fonction du mode, l'Amstrad permet d'afficher simultanément plusieurs couleurs sur l'écran :

Mode 0 : 16 couleurs parmi 27
Mode 1 : 4 couleurs parmi 27
Mode 2 : 2 couleurs parmi 27

Ces couleurs peuvent être obtenues en mode fixe (le troisième paramètre associé à la commande INK n'est pas précisé) ou en mode clignotant (le troisième paramètre associé à la commande INK est précisé).

EXEMPLE DE PROGRAMME

Le programme suivant illustre l'emploi de l'instruction INK ; trois boucles imbriquées (lignes 70 à 230) permettent d'afficher quelques lignes de texte et de faire varier simultanément les couleurs du bord de l'écran ainsi que celles du fond et des caractères.

```
10 MODE 0
20 INK 2,0
30 INK 3,0
40 PAPER 2
50 PEN 3
60 CLS
70 FOR I=0 TO 26
80 BORDER I
90 FOR I1=0 TO 26
100 INK 2,I1
110 FOR I2=0 TO 26
120 INK 3,I2
130 CLS
140 LOCATE 1,6
150 PRINT "COUL. BORD  : °";I;"§"
160 LOCATE 1,12
170 PRINT "COUL. FOND  : °";I1;"§"
180 LOCATE 1,18
190 PRINT "COUL. STYLO : °";I2;"§"
200 FOR J=1 TO 1000 : NEXT J
210 NEXT I2
220 NEXT I1
230 NEXT I
```

La fonction INKEY permet de savoir si une touche particulière du clavier a été frappée.

FORMAT

INKEY(numéro de touche)

- **numéro de touche** correspond à une touche quelconque du clavier ; par exemple, la touche "O" porte le numéro 34 et la touche "N" porte le numéro 46.

COMMENTAIRES

La fonction INKEY interroge le clavier 50 fois par seconde (la période est de 0,02 seconde) ; il faut lui indiquer le numéro de la touche à scruter. Si la touche n'a pas été frappée, la fonction retourne la valeur −1 ; si la touche a été frappée, elle retourne la valeur 0.

Par exemple, l'exécution d'un programme peut être interrompue tant que l'utilisateur n'a pas donné de réponse positive, c'est-à-dire tapé "O" ou "o". Deux solutions se présentent : regarder si le caractère tapé est un "o" minuscule ou un "O" majuscule, ou bien attendre que la touche correspondant aux deux caractères soit activée. Les lignes suivantes permettent d'interrompre le déroulement du programme tant que la touche n° 34 ("O") n'a pas été frappée :

```
50 PRINT "TAPEZ LA TOUCHE O POUR CONTINUER"
60 WHILE INKEY(34) : WEND
70 PRINT "SUITE !"
```

Le programme boucle sur la ligne 60 tant qu'aucune touche n'est frappée (il en est de même si la touche activée n'est pas la touche 34) ; la fonction retourne la valeur −1 (qui est une affirmation *vraie*) et la boucle WHILE est exécutée. Lorsque la touche "O" est frappée, la fonction retourne la valeur 0 (affirmation *fausse*), la boucle se termine et le programme passe à la ligne 70.



INKEY$

Fonction

La fonction INKEY$ lit un caractère frappé au clavier.

FORMAT

A$ = INKEY$

COMMENTAIRES

Le caractère frappé au clavier est lu par la fonction INKEY$ au moment où celle-ci est exécutée par le programme. Il est stocké dans une variable (ici A$) dont la valeur est aussitôt transmise au programme sans qu'il y ait ni affichage ni interruption de l'exécution. Un seul caractère peut être saisi à la fois par la fonction INKEY$. Lorsque aucun caractère n'est frappé au clavier alors que la fonction est exécutée, celle-ci retourne une chaîne vide dans la variable de stockage.

EXEMPLE

Le programme suivant effectue une boucle sans fin tant qu'une touche quelconque n'est pas pressée par l'utilisateur :

```
20 PRINT "Appuyez sur une touche quelconque pour commencer"
30 A$= " "
40 WHILE A$=" " : A$=INKEY$ : WEND
```

La structure d'attente aurait pu aussi être écrite de la manière suivante :

```
20 PRINT "Appuyez sur une touche quelconque pour commencer"
30 A$=INKEY$ : IF A$=" " THEN 30
...
```

La fonction INKEY$ est souvent utilisée dans une telle boucle, afin de laisser à l'utilisateur le temps d'appuyer sur une touche.

La fonction INP retourne la valeur de l'octet lu sur le port d'entrée/sortie spécifié. La fonction OUT adresse un octet de donnée sur le port d'entrée/sortie spécifié.

FORMATS

Y = INP(n° de port d'entrée/sortie)

OUT n° de port d'entrée/sortie, m

- Le **numéro de port** est un nombre entier compris entre 0 et 65535.
- **m** est une expression numérique entière comprise entre 0 et 255 qui détermine la valeur spécifiée pour l'octet de donnée.

COMMENTAIRES

Les fonctions INP et OUT s'emploient peu en BASIC. Le microprocesseur Z80 de l'Amstrad utilise en particulier les adresses d'entrée/sortie inférieures à &7FFFF pour gérer le transfert d'informations entre toutes les unités (clavier, écran, unités de disque, unité centrale, etc.). L'emploi des fonctions INP et OUT nécessite une connaissance précise de la partie matérielle (*hardware*) de l'ordinateur.

Signalons, à titre indicatif, que les adresses hexadécimales &F8xx, &F9xx, &FAxx et &FBxx sont disponibles pour les utilisateurs. Les caractères génériques xx peuvent prendre des valeurs comprises entre &DC et &DF pour les interfaces de communication et entre &E0 et &FE pour des périphériques quelconques.



Instruction

L'instruction INPUT permet d'entrer des données à partir du canal spécifié.

FORMAT

INPUT < # n° de canal, > < ''chaîne'' séparateur > liste de variables

- Le **numéro de canal** est un nombre compris entre 0 et 9.
- Le paramètre optionnel ''**chaîne**'' est une chaîne de caractères (qui doit être écrite entre guillemets) qui est affichée sur l'écran au moment où l'instruction INPUT est exécutée.
- **Séparateur** peut être une virgule ou un point-virgule ; la virgule entraîne seulement l'affichage du message entre guillemets, tandis que le point-virgule provoque l'affichage d'un point d'interrogation après le message.
- Les éléments de la **liste de variables** doivent être séparés par une virgule. Ces éléments peuvent être des variables de chaîne ou des variables numériques. Les valeurs entrées au clavier (ou lues dans un fichier) sont affectées aux variables en respectant l'ordre de la liste.

Exemples

```
INPUT "Valeur donnée à la variable X";X

INPUT "Entrer X et Y";X,Y

INPUT X

INPUT "Nom,age",A$,A

INPUT #1,A$,B$,A,B
```

COMMENTAIRES

Lorsqu'une instruction INPUT est rencontrée au cours de l'exécution d'un programme, celle-ci est suspendue en attente d'une entrée

de données par le canal spécifié. Si le paramètre optionnel ''chaîne'' est spécifié dans l'instruction, cette chaîne de caractères est affichée à l'écran, suivie d'un point d'interrogation. Dans le cas contraire, seul un point d'interrogation apparaît à l'écran au moment où l'instruction INPUT est exécutée. Les données entrées au clavier doivent être séparées par des virgules. L'ordre de leur affectation aux variables spécifiées dans l'instruction respecte l'ordre d'entrée (ou de lecture). Les valeurs entrées au clavier sont validées par un retour chariot qui doit être exécuté une fois que toutes les données de la liste ont été fournies. S'il est exécuté avant, le message :

```
?Redo from start
```

est affiché sur l'écran, signifiant que des données doivent être réintroduites. La liste des variables d'une instruction INPUT (ou INPUT#) n'est pas nécessairement homogène. Elle peut contenir des variables de types différents (voir exemples 4 et 5). Cependant, les valeurs entrées au clavier ou lues dans le fichier doivent correspondre aux types de variables auxquelles elles seront affectées. Les valeurs de chaîne entrées au clavier n'ont pas besoin d'être écrites entre guillemets, sauf si elles contiennent une virgule ou un espace significatif de début ou de fin.

Pour pouvoir entrer des données à partir d'un fichier (instruction INPUT#9), celui-ci doit avoir été au préalable ouvert (voir l'instruction OPENIN).



Fonction

La fonction INSTR recherche la première localisation d'une chaîne de caractères B$ dans une autre chaîne de caractères A$. Elle renvoie une valeur numérique correspondant à la position de B$ dans A$. Un paramètre facultatif, n, permet de commencer la recherche à un endroit quelconque à l'intérieur de la chaîne A$.

FORMAT

INSTR(< n, > A$,B$)

- **n** est une expression numérique entière comprise entre 1 et 255. Lorsque la valeur de ce paramètre facultatif n'est pas précisée, la valeur 1 est retenue par défaut, c'est-à-dire que la recherche commence à partir du premier caractère de A$.

Exemples

```
PRINT INSTR("AEIOUY","U")
```

Le résultat est 5.

```
PRINT INSTR("Jean-Marc","Marc")
```

Le résultat est 6.

```
PRINT INSTR(5,"Jean-Marc","Marc")
```

Le résultat est 6 (bien que la recherche commence à partir du 5e caractère, la valeur retournée correspond à la position dans la chaîne entière).

Pour cette raison, l'instruction :

```
PRINT INSTR(5,"Jean-Marc","a")
```

retournera la valeur 7, alors que l'instruction :

```
PRINT INSTR("Jean-Marc","a")
```

retournera la valeur 3.

RÈGLES DE SYNTAXE ET COMMENTAIRES

Lorsque B$ est une chaîne nulle, la fonction INSTR retourne la valeur n (ou 1, si n n'a pas été précisée). Lorsque la chaîne A$ ne contient pas B$, la fonction INSTR renvoie la valeur 0 ; il en est de même lorsque la chaîne A$ est nulle, ou lorsque n > LEN(A$), c'est-à-dire lorsque la valeur de n excède le nombre de caractères contenus dans la chaîne A$.

Un message d'erreur ''Improper argument'', argument incorrect, est affiché lorsque n est supérieur à 255.

La fonction JOY renvoie l'état du *joystick* (manette de jeu) spécifié.

FORMAT

JOY(n° de joystick)

- **numéro de joystick** peut, suivant le périphérique désiré, prendre la valeur 0 ou 1.

COMMENTAIRES

L'Amstrad scrute le clavier et les joysticks 50 fois par seconde. Il existe deux moyens de connaître l'état des manettes de jeu : la fonction JOY et la fonction INKEY.

Le tableau suivant indique les valeurs retournées par les fonctions suivant l'état des manettes :

Position de la manette 0	Fonction INKEY correspondante	Fonction JOY correspondante
Haut	INKEY(72) = 0	JOY(0) = 1
Bas	INKEY(73) = 0	JOY(0) = 2
Gauche	INKEY(74) = 0	JOY(0) = 4
Droite	INKEY(75) = 0	JOY(0) = 8
Fire 2	INKEY(76) = 0	JOY(0) = 16
Fire 1	INKEY(77) = 0	JOY(0) = 32

Par exemple, un sous-programme testant l'état de la manette de jeu n° 0 peut être écrit des deux manières suivantes :

```
50 IF JOY(0)=16 THEN GOSUB 1000
```

ou bien :

```
50 IF INKEY(76)=0 THEN GOSUB 1000
```

Si le bouton ''fire2'' du joystick n'est pas activé, la fonction JOY(0) renvoie la valeur 0, tandis que la fonction INKEY(76) retourne la valeur −1.

La commande KEY permet de redéfinir les touches de fonction du clavier.

FORMAT

KEY numéro de la touche,chaîne de caractères

- Le **numéro de la touche** doit être compris entre 128 et 159.
- Chaque **chaîne de caractères** ne doit pas comporter plus de 32 caractères et la somme des caractères ajoutés ne doit pas dépasser 100 caractères. Il peut s'agir d'une constante, d'une variable ou d'une expression exprimée sous forme de chaîne de caractères. Les expressions telles que les mots clés BASIC doivent être placées entre guillemets.

COMMENTAIRES

Lors de la redéfinition des touches de fonction, il est possible d'inclure, après la chaîne de caractères, le code d'un retour chariot, soit CHR$(13). Si la chaîne correspond à une commande BASIC, celle-ci sera alors exécutée dès que la touche de fonction sera pressée.

Exemples

```
KEY 135,"SAVE "+CHR$(34)
```

Lorsque la touche F7 sera pressée, le message :

```
SAVE "
```

s'affichera à l'écran. Il ne serait pas très astucieux d'inclure un retour chariot après cette instruction, celle-ci étant généralement utilisée en précisant le nom du programme qui doit être sauvegardé.

En revanche, si l'on veut redéfinir la touche F9 de sorte que celle-ci exécute la commande NEW (qui ne nécessite aucun argument particulier), il suffit de taper :

```
KEY 137,"NEW"+CHR$(13)
```

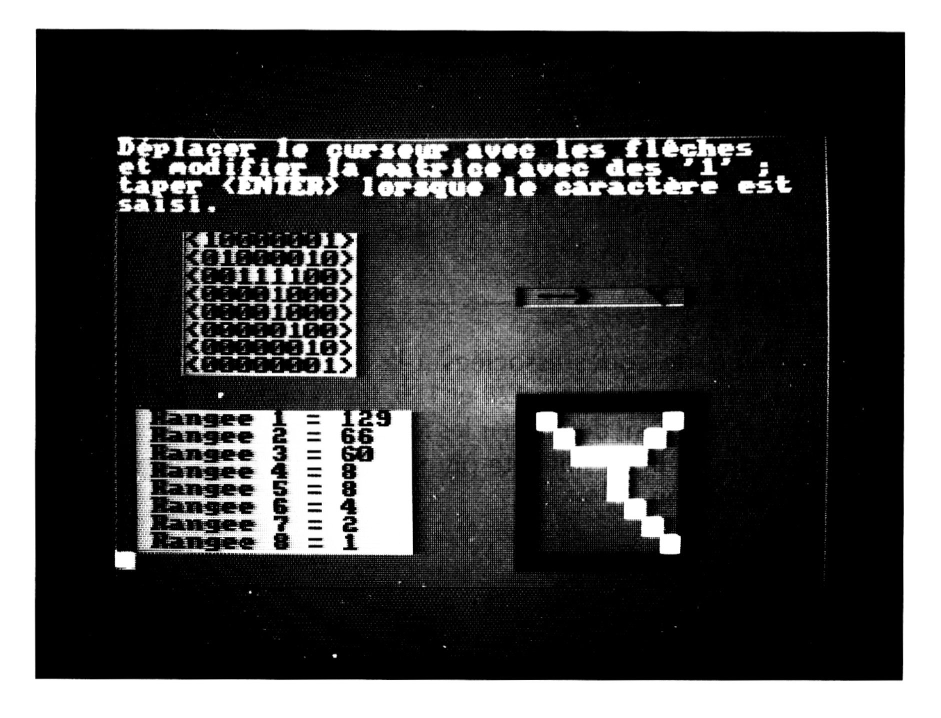
EXEMPLE DE PROGRAMME

Le programme ci-dessous permet de redéfinir des touches du clavier. A titre d'exemple, les touches fonction f1, f4, f7, f8 et f9 ont été reconfigurées pour représenter des lettres accentuées (lignes 50 à 150). La seconde partie du programme donne à l'utilisateur la possibilité de générer ses propres caractères en complétant une matrice dessinée sur l'écran. Lorsque la saisie est terminée, le programme affiche les valeurs décimales correspondant à chacun des huit octets codant le caractère.

```
10 MODE 1
20 BORDER 3
30 INK 0,1 : INK 1,24 : INK 2,9 : INK 3,
0
40 PAPER 0 : PEN 1
50 SYMBOL AFTER 130
60 SYMBOL 130,8,16,60,102,126,96,60,0
70 SYMBOL 133,32,16,120,12,124,204,118,0
80 SYMBOL 136,24,36,60,102,126,96,60,0
90 SYMBOL 138,16,8,60,102,126,96,60,0
100 SYMBOL 151,16,8,102,102,102,102,62,0
110 KEY 135,CHR$(138)
120 KEY 136,CHR$(130)
130 KEY 137,CHR$(136)
140 KEY 132,CHR$(133)
150 KEY 129,CHR$(151)
160 CLS
170 PRINT "Des lettres accentuees ont et
e affectees";
180 PRINT"aux touches de fonction f1, f4
, f7, f8"
190 PRINT "et f9."
200 PRINT
210 LOCATE 1,25
220 PRINT "(taper une touche pour contin
uer !)";
230 C$=""
240 WHILE C$="" : C$=INKEY$ : WEND
250 WHILE 1
260 CLS
```
→

```
270 PRINT "Deplacer le curseur avec les
fleches"
280 PRINT "et modifier la matrice avec d
es '1' ;"
290 PRINT "taper <ENTER> lorsque le cara
ctere est  saisi."
300 PAPER 2 : PEN 3
310 WINDOW #1,25,33,16,24
320 WINDOW #2,24,33,15,15
330 PAPER #2,3
340 CLS #2
350 WINDOW #3,24,33,24,24
360 PAPER #3,3
370 CLS #3
380 WINDOW #4,24,24,16,23
390 PAPER #4,3
400 CLS #4
410 WINDOW #5,33,33,16,23
420 PAPER #5,3
430 CLS #5
435 PEN #1,1
440 FOR I=1 TO 8
450 LOCATE 5,I+5
460 PRINT "<00000000>"
470 NEXT I
480 CURSOR 1,1
490 C$=""
500 WHILE C$<>CHR$(13)
510 LOCATE 6,6
520 WHILE POS(#0)>5 AND POS(#0)<14 AND V
POS(#0)>5 AND VPOS(#0)<14 AND C$<>CHR$(1
3)
530 C$=""
540 WHILE C$="" : C$=INKEY$ : WEND
550 IF C$="1" OR C$="0" THEN GOSUB 860 :
 PRINT C$;
560 IF C$=CHR$(240) THEN LOCATE POS(#0),
VPOS(#0)-1
570 IF C$=CHR$(241) THEN LOCATE POS(#0),
VPOS(#0)+1
580 IF C$=CHR$(242) THEN LOCATE POS(#0)-
1,VPOS(#0)
590 IF C$=CHR$(243) THEN LOCATE POS(#0)+
1,VPOS(#0)
600 WEND
610 WEND
620 FOR J=1 TO 8
630 LOCATE 2,15+J
640 PRINT SPACE$(16);
650 NEXT J
```

```
660 FOR J=1 TO 8
670 LIR$=""
680 FOR I=6 TO 13
690 LOCATE I,5+J
700 LIR$=LIR$+COPYCHR$(#0)
710 NEXT I
720 A(J)=VAL("&X"+LIR$)
730 LOCATE 3,J+15
740 PRINT "Rangee";J;"=";A(J)
750 NEXT J
760 LOCATE 24,9
770 PRINT " -->    ";
780 SYMBOL 240,A(1),A(2),A(3),A(4),A(5),
A(6),A(7),A(8)
790 PRINT CHR$(240);" ";
800 PAPER 0 : PEN 1
810 LOCATE 1,24
820 C$=""
830 WHILE C$="" : C$=INKEY$ : WEND
840 WEND
850 END
860 X=POS(#0) : Y=VPOS(#0)
870 LOCATE #1,X-5,Y-5
880 IF C$="1" THEN PRINT #1,CHR$(143); E
LSE PRINT #1,CHR$(128)
890 RETURN
```





Commande

La commande KEY DEF permet de modifier le caractère produit par la frappe d'une touche quelconque.

FORMAT

KEY DEF n° de touche, répétition < ,normal < ,avec SHIFT < ,avec CTRL > > >

- Le **numéro de touche** est une valeur comprise entre 0 et 79 ; elle correspond au numéro des touches du clavier et des joysticks.
- **répétition** peut prendre les valeurs 0 ou 1 ; la valeur 0 place la touche en mode non-répétition alors que la valeur 1 l'initialise en mode répétition.
- **normal** correspond au code ASCII du caractère qui sera affiché lorsque la touche sera activée seule.
- **SHIFT** correspond au code ASCII du caractère qui apparaîtra lorsque la touche sera frappée simultanément avec la touche SHIFT.
- **CTRL** correspond au code ASCII du caractère qui sera affiché lorsque la touche sera frappée simultanément avec la touche CTRL.

COMMENTAIRES

Si les valeurs de codes associées à la commande KEY DEF sont comprises entre les nombres 128 et 159, les caractères qui apparaissent sont ceux qui sont définis par les fonctions KEY correspondantes.

Par exemple, la ligne suivante permet de redéfinir la touche TAB en mode répétition :

```
KEY DEF 68,1,65,145,176
```

La frappe de la touche TAB seule provoque l'affichage du caractère A, la frappe simultanée des touches TAB et SHIFT n'a aucun effet

et la frappe simultanée des touches TAB et CTRL génère la lettre alpha.
La commande :

```
KEY 145,"A ou alpha"
```

attribue une valeur au caractère étendu de code 145. Maintenant, la frappe des touches TAB et SHIFT entraîne l'affichage :

```
A ou alpha
```

LEFT$ et RIGHT$

Les fonctions LEFT$ et RIGHT$ extraient des caractères en certaines positions d'une chaîne ; LEFT$ fournit les n caractères de gauche, tandis que la fonction RIGHT$ donne les n caractères de droite de la chaîne.

FORMATS

LEFT$(a$,n)

RIGHT$(a$,n)

- **a$** est une expression chaîne de caractères.
- **n** est une expression numérique entière (comprise entre 0 et 255) spécifiant le nombre de caractères qui doivent être extraits.

Exemples

```
A$=LEFT$("BASIC AMSTRAD",5)
B$=RIGHT$("BASIC AMSTRAD",7)
```

Dans cet exemple, A$ et B$ prennent respectivement pour valeur "BASIC" et "AMSTRAD".

COMMENTAIRES

Si la valeur de n est supérieure au nombre de caractères de la chaîne, celle-ci est fournie en entier. Si n est égal à 0, on obtient une chaîne nulle (c'est-à-dire de longueur 0).

La fonction MID$ permet d'extraire des caractères situés en n'importe quelle position d'une chaîne (voir la description de cette fonction).

Cet exemple constitue la seconde partie du programme donné en illustration de l'instruction FOR/NEXT. Après qu'une série de résultats de rencontres sportives entre quatre équipes a été présentée sous la forme d'un tableau à deux entrées, il s'agit d'établir le classement de ces quatre équipes en fonction des résultats obtenus. Une équipe ayant remporté une rencontre marque deux points, celle qui l'a perdue n'en marquant pas. En cas de match nul, chacune des deux équipes marque un point. Le classement ci-dessous suppose que les résultats ont été entrés dans le format x – y, x et y étant des nombres inférieurs à 10.

Le résultat de la rencontre entre l'équipe A$(I) et l'équipe A$(J) est stocké dans la variable B$(I,J). Chaque chaîne de caractères B$(I,J) commence par un espace qui a été ajouté au résultat (ligne 70) afin de le centrer dans le tableau. Le format d'une chaîne de résultat est donc le suivant : *x – y, l'astérisque représentant un espace. La première étape (ligne 290) consiste à supprimer cet espace, ce qui est réalisé au moyen de la fonction RIGHT$(B$(I,J),3).

Il s'agit ensuite de comparer x à y : lorsque x > y, l'équipe A$(I) marque deux points, qui sont comptabilisés dans la variable numérique A(I). Lorsque x < y, les deux points sont marqués par l'équipe A$(J) et comptabilisés dans la variable numérique A(J). En cas de match nul, chaque équipe A$(I) et A$(J) marque un point comptabilisé dans les variables numériques correspondantes. Pour pouvoir comparer ainsi x à y, il faut pouvoir les extraire de la chaîne de résultat B$(I,J). La fonction LEFT$(B$(I,J),1) permet d'extraire x et la fonction RIGHT$(B$(I,J),1) permet d'extraire y. Ces fonctions renvoient des valeurs de chaîne (bien que x et y soient ici des nombres). Lorsque l'on écrit x < y et que x et y sont des nombres extraits par les fonctions LEFT$ et RIGHT$, on compare en fait des chaînes de caractères. Cette comparaison s'effectue en fonction des codes ASCII (voir la fonction MID$). Les numéros de code des chiffres étant classés dans l'ordre de leur valeur numérique, il est donc indifférent de comparer ces chiffres sous forme de chaînes ou sous forme de valeurs numériques.

Une fois que les points de chaque rencontre ont été attribués aux différentes équipes (lignes 270 à 320), le classement est établi selon la technique de tri dichotomique effectué sur les variables numériques A(I). Le processus est décrit à propos de l'instruction MID$.



Fonction

La fonction LEN fournit le nombre de caractères contenus dans une chaîne spécifiée comme argument.

FORMAT

LEN(a$)

- **a$** est une expression chaîne de caractères.

COMMENTAIRES

La fonction LEN fournit une valeur numérique. Tous les caractères de la chaîne a$ sont comptés, y compris les espaces éventuels.

Exemples

```
PRINT LEN("BASIC AMSTRAD")
13

A$="LEN signifie LENGTH, longueur"
PRINT LEN(A$)
29
```

(Ces exemples sont exécutables en mode programme ou en mode direct).

L'instruction LET permet d'assigner la valeur d'une expression à une variable. Le mot LET est facultatif dans la procédure d'assignation.

FORMAT

< LET > variable = expression

- **variable** est le nom de la variable ou l'élément du tableau auquel une valeur doit être affectée. Il peut s'agir d'une variable de chaîne ou d'une variable numérique.
- **expression** est l'expression (chaîne de caractères ou expression numérique) dont la valeur doit être affectée à la variable.

COMMENTAIRES

Le type de l'expression (numérique ou chaîne de caractères) doit correspondre à celui de la variable. Dans le cas contraire, un message d'erreur "Type mismatch" (non-concordance de type) est affiché.

Le mot LET est facultatif : le signe égal (=) suffit à affecter la valeur d'une expression à une variable. Dans tous les exemples donnés dans cet ouvrage, le mot LET est omis dans les procédures d'assignation (la syntaxe est ainsi plus proche de celle qui est employée en algèbre).

Par exemple, dans les trois lignes de programme suivantes :

```
50 LET LANGAGE$="BASIC"
60 LET VERS$=" AMSTRAD 1985"
70 LET NOM$=LANGAGE$+VERS$
```

le mot LET peut chaque fois être supprimé. La valeur "BASIC" est assignée à la variable de chaîne LANGAGE$. La valeur "AMSTRAD 1985" est assignée à la variable de chaîne VERS$. Il est à noter que la valeur en question contient des chiffres. Il s'agit néanmoins d'une valeur de chaîne, puisqu'elle est écrite entre guillemets. La somme des valeurs des variables LANGAGE$ et VERS$ est ensuite assignée

à la troisième variable de chaîne NOM$. Celle-ci prend donc la valeur ''BASIC AMSTRAD 1985''.

Contrairement à d'autres BASIC, celui de l'Amstrad permet que le nom donné à une variable comporte tout ou partie d'un mot clé BASIC. Par exemple, VERSION$ est permis bien qu'il contienne le mot clé ON. Un nom de variable doit toujours commencer par une lettre.

Instruction

LINE INPUT

Cette instruction permet de donner pour valeur à une variable de chaîne une suite de caractères entrés au clavier ou lus dans un fichier.

FORMATS

```
LINE INPUT < #numéro de canal, > < ; >
           < message, séparateur > < variable de chaîne >
```

- Le **numéro de canal** doit être compris entre 0 et 9. La valeur par défaut est 0 ; elle correspond à une entrée clavier.
- Le **message** est un texte (optionnel) qui est affiché à l'écran lorsque l'instruction LINE INPUT est exécutée. Si le séparateur est un point-virgule, le message est suivi d'un point d'interrogation ; si le séparateur est une virgule, le curseur est placé immédiatement après le message.
- **variable de chaîne** est le nom de la variable à laquelle la valeur (c'est-à-dire la suite de caractères) entrée au clavier ou lue dans un fichier (si le numéro de canal est égal à 9) sera affectée.

COMMENTAIRES

Tous les caractères (au maximum 254) entrés au clavier, ou lus dans le fichier une fois l'instruction LINE INPUT exécutée, sont considérés comme faisant partie de la chaîne, jusqu'à ce qu'un retour chariot soit exécuté (dans le cas d'une entrée clavier) ou que son code (0D) soit rencontré au cours de la lecture dans le fichier.

Tous les caractères (excepté le retour chariot) étant considérés comme faisant partie de la chaîne, les virgules, qui sont reconnues comme séparateurs de l'instruction INPUT, ne le sont pas dans le cas d'une instruction LINE INPUT.

Exemple

```
10 DIM A$(12)
20 FOR I=1 TO 12
30 LINE INPUT "Nom,Prénom,Téléphone ";A$(I)
NEXT I
```

Une fois ce programme lancé, le message :

```
Nom,Prénom,Téléphone ?
```

est affiché douze fois de suite, chaque réponse entrée au clavier étant donnée comme valeur à la variable correspondante. Ces réponses peuvent être entrées dans n'importe quel format (avec ou sans virgule, réponses complètes ou incomplètes, etc.). Après chaque réponse, un retour chariot doit être tapé, signifiant à l'ordinateur que les caractères entrés depuis le précédent retour chariot constituent la chaîne à donner comme valeur à la variable A$(I) en cours.

Dans le programme illustrant les instructions OPENIN et OPENOUT, les lignes 280 à 330 permettent de lire le contenu du fichier d'adresses ; à chaque réitération de la boucle, la ligne 320 assigne une chaîne de caractères à la variable T$(INC%) ; la chaîne de caractères se termine lorsqu'un caractère retour chariot (CHR$(13)) est rencontré.

La commande LIST affiche à l'écran ou sur l'imprimante la liste (le *listing*) des lignes de programme se trouvant en mémoire centrale de l'ordinateur.

FORMAT

LIST < ligne1 > < – < ligne2 > > < ,#N° de canal >

- **ligne1** est le numéro de la première ligne à afficher.
- **ligne2** est le numéro de la dernière ligne à afficher.
- Le **numéro de canal** désigne le canal sur lequel doit être sorti le listing. La valeur par défaut est 0, ce qui correspond à la fenêtre 0 de l'écran. Pour éditer le listing sur l'imprimante, le canal 8 doit être sélectionné.

RÈGLES DE SYNTAXE ET COMMENTAIRES

Après que la commande LIST a été exécutée, l'ordinateur revient en mode direct. L'affichage du listing (c'est-à-dire de la liste des lignes d'instructions qui composent le programme) peut être interrompu à tout moment en appuyant une ou deux fois sur la touche ESC. Dans le premier cas, le défilement du listing pourra être repris en appuyant sur n'importe quelle touche (excepté ESC). Selon le format utilisé, la commande LIST a les effets suivants (les exemples donnés ici pour la sortie écran s'appliquent aussi à la sortie imprimante, en sélectionnant le canal 8) :

```
LIST
```

liste tout le programme en mémoire, de la première à la dernière ligne.

```
LIST m
```

affiche à l'écran la ligne de numéro m.

```
LIST m-
```

liste le programme de la ligne m jusqu'à la fin.

```
LIST -n
```

liste le programme de la première à la ligne n incluse.

```
LIST m-n
```

liste toutes les lignes du programme en mémoire dont les numéros sont compris entre n et m (lignes incluses).

Le programme donné en illustration de l'instruction WINDOW montre comment l'on peut afficher un listing sur une fenêtre particulière de l'écran.

L'instruction LOAD charge en mémoire centrale de l'ordinateur un programme ou fichier sauvegardé sur cassette ou sur disquette au moyen de l'instruction SAVE.

FORMAT

LOAD ''spécification du fichier'' < ,adresse >

- L'argument **''spécification du fichier''** doit respecter les règles décrites à propos de l'instruction SAVE et dans la rubrique ''Commandes AMSDOS''.
- L'option **adresse** est utilisée avec les fichiers binaires ; elle représente l'adresse à partir de laquelle le fichier doit être chargé. Si celle-ci n'est pas mentionnée, il est chargé à l'adresse qu'il occupait lors de sa sauvegarde.

COMMENTAIRES

Lorsque le chargement doit s'effectuer à partir d'une cassette et que le nom du fichier n'est pas précisé, le premier programme trouvé sur la bande magnétique est chargé en mémoire centrale.

Avant de charger un programme, LOAD ferme tous les fichiers ouverts et efface toutes les données et les programmes utilisateur résidant en mémoire centrale.

Si le programme a été sauvegardé avec l'option P, il ne peut pas être chargé avec la commande LOAD. Il faut utiliser la commande RUN.



Instruction

L'instruction LOCATE positionne le curseur sur l'écran, en mode texte.

FORMAT

LOCATE < numéro de canal, > coord x, coord y

- Le **numéro de canal** correspond à une fenêtre définie avec l'instruction WINDOW ; il peut prendre des valeurs comprises entre 0 et 7. La valeur par défaut est 0.
- **x** et **y** définissent respectivement les nouvelles coordonnées horizontale et verticale du curseur ; ces valeurs sont relatives à la fenêtre spécifiée.

COMMENTAIRES

Les lignes de l'écran sont numérotées de 1 à 25, de haut en bas de celui-ci ; de la même façon, les colonnes sont numérotées à partir de 1 (position correspondant au bord gauche de l'écran).

Les fonctions POS et VPOS retournent les numéros de ligne et de colonne correspondant à la position courante du curseur.

EXEMPLE DE PROGRAMME

Le programme de gestion de carnet d'adresses (instructions OPENIN et OPENOUT) utilise l'instruction LOCATE pour présenter l'affichage des menus et des fiches à l'écran.

Les fonctions LOWER$ et UPPER$ permettent respectivement de convertir les lettres d'une chaîne de caractères en minuscules ou en majuscules.

FORMATS

LOWER$ (chaîne de caractères)

UPPER$ (chaîne de caractères)

- **chaîne de caractères** est une variable alphanumérique quelconque.

COMMENTAIRES

Les instructions suivantes permettent de tester si la lettre "O" est tapée en majuscule ou en minuscule :

```
50 C$=""
60 WHILE C$="" : C$=INKEY$ : WEND
70 IF UPPER$(C$)="O" THEN GOSUB 1000
...
```

La ligne 70 aurait pu également être écrite de la manière suivante :

```
70 IF LOWER$(C$)="o" THEN GOSUB 1000
```



Instruction
(CPC 664 et CPC 6128)

L'instruction MASK permet de tracer des traits en pointillé suivant une trame définie.

FORMAT

MASK < expression entière > < ,état du 1er point >

- L'**expression entière** est un nombre compris entre 0 et 255.
- L'**état du 1er point** peut prendre les valeurs 0 (tracé) ou 1 (non tracé).

COMMENTAIRES

La commande MASK définit l'état des groupes de huit pixels qui seront tracés par les instructions DRAW et DRAWR ultérieures ; ce modèle correspond à la valeur binaire de l'expression (dans laquelle les chiffres 1 représentent des pixels allumés tandis que les chiffres 0 représentent des pixels devant rester éteints). Par exemple, les instructions suivantes allument un point sur deux :

```
10 MASK &X10101010,1
20 MOVE 320,0
30 DRAW 320,400
```

Pour tracer une ligne en trait plein, il faut modifier l'instruction MASK de la manière suivante :

```
10 MASK &X11111111,1
```

EXEMPLE DE PROGRAMME

Pour visualiser l'effet de la commande MASK, il est possible d'ajouter une ligne d'instruction au programme illustrant les commandes DRAW et DRAWR :

```
305 MASK &X11001100,0
```

Le programme trace maintenant la sinusoïde amortie en pointillé.

L'instruction MEMORY permet de modifier la taille de la mémoire RAM utilisable en BASIC.

FORMAT

MEMORY adresse

- L'**adresse** ne doit pas être supérieure à 43903 (CPC 464) ou 42619 (CPC 664), valeur maximale imposée à l'initialisation du système.

COMMENTAIRES

L'adresse la plus haute utilisable en BASIC peut être connue au moyen de la variable HIMEM (voir la description de ce mot clé). Les chiffres retournés par cette variable sont inférieurs (de 4K) à ceux qui sont donnés ci-dessus en cas d'utilisation d'un fichier tampon.

Deux raisons principales peuvent justifier l'abaissement de la mémoire haute disponible en BASIC :

- la nécessité de réserver une partie de la mémoire RAM pour y implanter des programmes utilisateur écrits en langage machine ;
- la possibilité donnée par l'Amstrad de porter de 16 à 32K la mémoire écran. Les 16K supplémentaires doivent être soustraits des 42K de mémoire RAM normalement accessibles en BASIC.

Le programme ci-après illustre cette dernière possibilité.

EXEMPLE DE PROGRAMME

Ce programme réalise une animation simple en affichant alternativement à l'écran deux dessins complémentaires stockés dans deux blocs différents de la mémoire écran. Il s'agit du dessin d'une roue, les deux motifs se superposant avec un léger décalage au niveau des

rayons. Ce décalage simule, lorsque l'on passe d'une mémoire à l'autre, le mouvement de la roue.

```
5 CLS:CLG
10 GOSUB 1000
20 ECRAN=ECRAN XOR &80: GOSUB 2000
30 GOTO 20
999 '
1000 '********************************
1001 '     INITIALISATION
1002 '********************************
1010 MEMORY 16383
1020 FOR J=0 TO 100
1030 READ X
1040 IF X=-1 THEN 1100
1050 POKE &8000+J,X
1060 NEXT J
1070 DATA &3E,&C0,&CD,&8,&BC,&C9
1100 ECRAN=&C0: GOSUB 2000
1110 DEBUT=1: GOSUB 1900
1180 DATA -1
1200 ECRAN=&40: GOSUB 2000
1210 DEBUT=5: GOSUB 1900
1299 RETURN
1895 '
1896 '********************************
1897 '       TRACE DES ROUES
1898 '********************************
1900 CLG:DEG
1910 ORIGIN 320,200: DEG
1920 FOR J=1 TO 360
1930 PLOT 100*COS(J),100*SIN(J),1
1940 NEXT J
1950 FOR J=DEBUT TO DEBUT+360 STEP 8
1960 MOVE 0,0
1970 DRAW 100*COS(J),100*SIN(J)
1980 NEXT J
1990 RETURN
1999 '
2000 '********************************
2001 '      CHANGEMENT D'ECRAN
2002 '********************************
2010 POKE &8001,ECRAN
2020 CALL &8000
2030 RETURN
```

Les instructions des lignes 1020 à 1070 correspondent à l'appel d'un sous-programme (en langage machine) du système d'exploitation. Elles

permettent de charger la valeur &C0 (adresse supérieure de la mémoire écran) dans le registre A du microprocesseur et appellent le sous-programme du système d'exploitation se trouvant à l'adresse &BC08. Ce sous-programme se charge de positionner la nouvelle adresse de la mémoire écran ainsi définie. En langage machine, ces instructions s'écriraient :

```
3EC0      LD      A,C0
CDBC08    CALL    BC08
C9        RET
```

Rappelons que l'architecture du microprocesseur Z80 nécessite d'entrer une valeur stockée sur deux octets en mettant la partie de poids faible du nombre dans l'octet d'adresse inférieure (8H doit donc être entré avant BCH, ligne 1070).

Le tracé des roues est effectué au moyen des fonctions SIN et COS. La modification de la valeur initiale de la variable DEBUT permet de décaler la place des rayons, sur les deux dessins.

La commande MERGE permet de fusionner un programme se trouvant sur disque ou sur cassette avec le programme résidant en mémoire centrale de l'ordinateur.

FORMAT

MERGE ''spécification du fichier''

- Les règles concernant les noms figurant dans l'argument **''spécification du fichier''** sont les mêmes que celles qui sont décrites à propos de l'instruction SAVE (voir également la rubrique ''Commandes AMSDOS'').

COMMENTAIRES

Le programme se trouvant sur l'unité périphérique (cassette ou disquette) est fusionné avec le programme résidant en mémoire centrale de l'ordinateur. Si des lignes portent le même numéro, les anciennes lignes sont remplacées par les nouvelles.

Lorsque le nom du fichier se trouvant sur cassette n'est pas précisé, le programme en mémoire centrale est fusionné avec le premier programme rencontré sur la bande magnétique.

Lors d'une fusion avec un programme stocké sur disquette, si le nom n'est pas spécifié, un message d'erreur est affiché ; il en est de même si le programme n'existe pas.

Si le programme est protégé (sauvegardé avec un paramètre P), il ne peut pas être fusionné avec le programme en cours.

Après exécution de la commande MERGE, BASIC revient en mode direct.



EXEMPLE

Supposons que le programme donné en exemple dans la rubrique ''Opérateurs logiques'' ait été stocké sous le nom CARTES1 au moyen de l'instruction suivante :

```
SAVE "CARTES1"
```

et que les lignes de programme complémentaires données comme illustration de l'instruction ON BREAK aient été stockées sous le nom CARTES2 (ces deux programmes n'ont aucune ligne qui se recouvre).

Pour fusionner les programmes CARTES1 et CARTES2 en mémoire centrale de l'ordinateur, les instructions ou commandes suivantes devront être exécutées :

```
LOAD "CARTES1"
MERGE "CARTES2"
```

Il suffira alors d'exécuter la commande RUN pour lancer les programmes fusionnés.



La fonction MID$ fournit une partie spécifiée d'une chaîne de caractères. En tant qu'instruction, MID$ remplace tout ou partie d'une chaîne par une autre chaîne (ou par une partie d'une autre chaîne).

FORMATS

- **A$, B$** et **C$** sont des chaînes de caractères,
- **n** et **m** sont des expressions numériques entières dont les valeurs sont comprises entre 1 et 255.

Fonction

MID$(A$,n < ,m >)

Exemples

```
MID$("Jean-Pierre",6)
```

Le résultat est "Pierre".

```
MID$("Jean-Pierre",1,4)
```

Le résultat est "Jean".

Instruction

MID$(B$,n < ,m >) = C$

Exemple

```
B$="Louis 11"
MID$(B$,7,2)="14"
PRINT B$
```

Le résultat est "Louis 14".

RÈGLES DE SYNTAXE ET COMMENTAIRES

Fonction

La fonction MID$ extrait de A$ une chaîne de m caractères, à compter du $n^{ième}$ caractère à partir de la gauche. Lorsque m est omis ou lorsque sa valeur est supérieure au nombre de caractères de A$ situés à droite du $n^{ième}$, tous les caractères au-delà du $n^{ième}$ sont fournis. Si m = 0 ou si n est supérieur au nombre de caractères de la chaîne A$, la fonction MID$ fournit une chaîne nulle.

Instruction

m caractères de la chaîne B$ sont remplacés, à partir de la position n, par un nombre équivalent de caractères appartenant à C$. L'argument m est facultatif. S'il n'est pas fourni, tous les caractères de la chaîne C$ sont utilisés. Cependant, que m soit ou non spécifié, la longueur de la chaîne B$ n'est pas modifiée après l'opération. Si le nombre de caractères à remplacer dans B$ est inférieur au nombre de caractères de C$, c'est-à-dire si m est supérieur à LEN(B$) – n, seuls les premiers caractères de C$ sont pris en compte pour le remplacement. Lorsque la valeur de m ou de n est hors limite, un message d'erreur ''Illegal function call'', appel fonction interdit, est affiché.

EXEMPLE DE PROGRAMME

Le programme ci-après classe par ordre alphabétique une suite de mots ou d'expressions (jusqu'à 100) entrés au clavier et stockés dans les variables de chaîne A$(I). Le classement est établi au moyen de la technique de tri dichotomique, c'est-à-dire que les éléments (mots ou expressions) sont comparés deux à deux jusqu'à épuisement de la liste. La comparaison de deux chaînes s'effectue caractère par caractère, au moyen de l'opérateur relationnel > , le classement étant fonction des numéros de code ASCII des caractères comparés (la liste des numéros de code ASCII est indiquée en annexe du guide de l'utilisateur livré avec l'Amstrad).

Utilisation des opérateurs relationnels sur des chaînes de caractères

Les opérateurs relationnels comparent deux valeurs qui peuvent être soit toutes deux numériques, soit toutes deux des chaînes de

caractères. Le résultat de la comparaison est *vrai* ou *faux*. Ce résultat est habituellement utilisé pour prendre une décision quant au déroulement du programme (au moyen d'une instruction IF...THEN...ELSE par exemple). Les opérateurs relationnels sont les suivants :

=	égal
< >	inégal
<	inférieur à
>	supérieur à
< = ou = <	inférieur ou égal à
> = ou = >	supérieur ou égal à

L'utilisation des opérateurs relationnels sur des chaînes de caractères sera illustrée à propos de l'opérateur < .

Une chaîne de caractères est inférieure à une autre si elle se place alphabétiquement avant celle-ci, en tenant compte des règles suivantes :

- Dès que le code ASCII du $n^{\text{ième}}$ caractère comparé (en partant de la gauche) diffère, la chaîne dont le caractère en question a le code le plus bas est considérée comme inférieure ; par exemple, ''Dupont'' est inférieur à ''Durand'' car ''p'' < ''r''.
- Si, lors de la comparaison, la fin d'une chaîne est rencontrée, cette chaîne est considérée comme inférieure à l'autre ; par exemple, ''Jean'' est inférieur à ''Jean-Pierre''.
- Les blancs, qu'ils soient en début, en milieu ou en fin de chaîne, sont significatifs et sont pris en compte, à ce titre, dans le classement. En conséquence, une chaîne commençant par un blanc sera inférieure à toute autre chaîne commençant par un autre caractère (car le caractère '' '' a le code ASCII le plus bas parmi les caractères imprimables, à savoir 32).

D'autre part, l'examen du tableau des codes ASCII donné en annexe du guide l'utilisateur montre que :

- Le numéro de code ASCII des minuscules est supérieur à celui des majuscules correspondantes ; par exemple, ''Henri'' est inférieur à ''alfred'' car ''H'' < ''a''.

- Les chiffres sont inférieurs aux lettres.
- Les minuscules accentuées ont des codes ASCII spécifiques, différents de ceux des majuscules et des minuscules non accentuées correspondantes. Cependant, ces caractères ne sont pas préprogrammés sur l'Amstrad. Leur code est fonction de l'emplacement choisi par l'utilisateur (voir l'instruction SYMBOL). Aussi supposerons-nous ici que les mots ou expressions à classer ne contiennent pas de minuscules accentuées.

Algorithme de tri de chaînes de caractères

Les considérations précédentes montrent que, pour classer par ordre alphabétique un ensemble de mots ou d'expressions, il ne suffit pas de comparer directement ces chaînes de caractères au moyen des opérateurs relationnels. Il est nécessaire d'effectuer au préalable un certain nombre d'opérations sur ces chaînes afin que le classement :

- ne différencie pas les majuscules des minuscules ;
- ne tienne pas compte des caractères de ponctuation tels que tirets, blancs (espacements) ou apostrophes ;

Le programme proposé effectue successivement les cinq opérations suivantes :

1. Entrée des chaînes de caractères au clavier et stockage dans les variables A$(I) (ligne 110). Une copie de chacune de ces chaînes est ensuite chargée dans les variables C$(I) afin d'être traitée ultérieurement, tout en conservant dans A$(I) les chaînes originales (ligne 170).
2. Élimination, dans chaque chaîne C$(I), des blancs, des apostrophes et des tirets (lignes 170 à 200).
3. Conversion des minuscules en majuscules (lignes 250 à 280).
4. Classement des chaînes C$(I) modifiées (lignes 330 à 350).
5. Affichage à l'écran des chaînes A$(I), selon le classement établi sur les chaînes C$(I) correspondantes.

La deuxième étape est décrite en tant qu'illustration de l'instruction IF...THEN...ELSE. La troisième étape, qui fait appel au mot clé MID$

en tant que fonction et instruction, et la quatrième étape sont explicitées à la suite du listing du programme.

```
10 DIM A$(100),C$(100)
20 CLS
30 '
40 '***********************************
50 '        ENTREE DES DONNEES
60 '***********************************
70 PRINT "Combien de mots ou d'expressio
ns"
80 INPUT "souhaitez-vous classer ";M
90 CLS
100 FOR I=1 TO M
110 PRINT "Nom numero ";I:INPUT A$(I)
120 NEXT I
130 '
140 '***********************************
150 ' ELIMINATION DES BLANCS, DES
      APOSTROPHES ET DES TIRETS
160 '***********************************
170 FOR I=1 TO M:C$(I)=A$(I):B$=" "
180 N=INSTR(1,C$(I),B$)
190 IF N=0 THEN IF B$=" " THEN B$="'" :
    GOTO 180 ELSE IF B$="'" THEN B$="-":
    GOTO 180 ELSE 200 ELSE C$(I)=LEFT$(
    C$(I),N-1) + MID$(C$(I),N+1)
200 IF N>0 THEN 180 ELSE NEXT I
210 '
220 '***********************************
230 '   CONVERSION DES MINUSCULES EN
                MAJUSCULES
240 '***********************************
250 FOR I=1 TO M : FOR J=1 TO LEN(C$(I))
260 V$=MID$(C$(I),J,1)
270 IF V$>="a" AND V$<="z" THEN W=ASC(
    V$)-32 : W$=CHR$(W) : MID$(C$(I),J)=
    W$
280 NEXT J,I
290 '
300 '***********************************
310 '     CLASSEMENT DES CHAINES
320 '***********************************
330 FOR I=1 TO M-1 : FOR J=I+1 TO M
340 IF C$(I)>C$(J) THEN D$=C$(I) : C$(I)
    =C$(J) : C$(J)=D$ : D$=A$(I) : A$(I)
    =A$(J) : A$(J)=D$
350 NEXT J,I
360 '
```

→

```
370 '************************************
380 '          SORTIE DES RESULTATS
390 '************************************
400 PRINT
410 FOR I=1 TO M : PRINT A$(I) : NEXT I
420 END
```

Conversion des minuscules en majuscules correspondantes (lignes 250 à 280)

Cette partie de programme aurait pu avantageusement faire appel à la fonction UPPER$. Celle-ci n'est pas employée ici uniquement dans un but didactique, afin d'une part d'illustrer la fonction MID$ et de montrer d'autre part au lecteur qu'il est souvent relativement simple d'écrire quelques lignes de programme destinées à remplacer une instruction n'existant pas dans le langage employé. Les fonctions UPPER$ et LOWER$ sont par exemple relativement spécifiques au BASIC Amstrad (elles existent en revanche dans d'autres langages évolués tels que Pascal ou C).

Ligne 260

MID$ est utilisé ici comme fonction ; elle fournit à la variable v$ le J^ième^ caractère, à partir de la gauche, de la chaîne C$(I).

Ligne 270

L'expression qui suit IF dans l'instruction IF...THEN est un exemple d'utilisation de l'opérateur logique AND dans une décision : l'expression est *vraie* (et la clause THEN est alors exécutée) si les deux conditions suivantes sont simultanément réalisées :

- v$ > = "a", c'est-à-dire que le code ASCII du caractère contenu dans la variable v$ est supérieur ou égal à celui de la lettre minuscule a, soit 097.
- v$ < = "z", c'est-à-dire que le code ASCII du caractère contenu dans la variable v$ est inférieur ou égal à celui de la lettre minuscule z, soit 122.

L'expression est donc *vraie* si le code ASCII du caractère contenu dans la variable v$ est compris entre 97 et 122, c'est-à-dire s'il s'agit d'une lettre minuscule (non accentuée). Dans ce cas (et seulement dans ce cas), les trois instructions de la clause THEN sont exécutées :

```
W=ASC(v$)-32
```

La valeur 32 est soustraite de celle du numéro de code ASCII correspondant à la minuscule contenue dans v$, et le résultat est stocké dans la variable numérique W. La valeur 32 représente la différence des numéros de code ASCII d'une minuscule non accentuée et de la majuscule correspondante (voir le tableau des codes ASCII dans le guide de l'utilisateur). La variable W contient ainsi le numéro de code ASCII de la majuscule correspondant à la minuscule contenue dans v$.

```
W$=CHR$(W)
```

Le caractère ayant W comme numéro de code ASCII, est stocké dans la variable de chaîne W$. Cette dernière contient donc la lettre majuscule correspondant à la minuscule contenue dans v$.

```
MID$(C$(I),J)=W$
```

MID$ est utilisé ici comme instruction. Le J^{ième} caractère de C$(I), à partir de la gauche, est remplacé par le premier (et ici unique) caractère contenu dans la variable W$. Cette opération a donc pour effet de remplacer la minuscule en J^{ième} position dans la chaîne C$(I) par la majuscule correspondante.

Lignes 250 à 280

Deux boucles FOR/NEXT permettent de répéter la suite d'instructions des lignes 260 et 270 pour chacun des caractères (du premier, J=1, au dernier, J=LENC$(I)) de chacune des chaînes C$(I) (de I=1 jusqu'à I=M, M étant le nombre total de mots ou d'expressions à classer).

Classement des chaînes modifiées C$(I) (lignes 330 à 350)

Les deux boucles FOR/NEXT imbriquées permettent de comparer la chaîne C$(1) avec la chaîne C$(2), puis avec la chaîne C$(3), etc. jusqu'à la dernière chaîne C$(M). Lors de chaque comparaison, la variable D$ est utilisée comme variable intermédiaire. Les chaînes I et J sont interverties lorsque la chaîne d'indice I est supérieure à la chaîne d'indice J. Les chaînes correspondantes non modifiées A$(I) sont interverties de la même manière.

MID$ est également utilisé comme fonction dans l'opération de concaténation de la dernière clause ELSE de la ligne 190. Les chaînes C$(I), telles qu'elles ont été modifiées pour établir le classement, peuvent également être affichées en regard des chaînes A$(I) correspondantes, en remplaçant, à la ligne 410, l'instruction PRINT A$(I) par l'instruction PRINT A$(I),C$(I).



L'opérateur MOD (modulo) donne la valeur entière du reste de la division d'un nombre entier par un autre nombre entier.

FORMAT

n MOD m

- **n** et **m** sont des expressions numériques que le BASIC AMSTRAD convertit au besoin en nombres entiers avant d'effectuer l'opération (arrondissant à la valeur entière la plus proche).

COMMENTAIRES

MOD est un opérateur arithmétique, au même titre que les opérateurs d'addition, de multiplication, etc. Le modulo se place, dans la hiérarchie des opérations arithmétiques, entre l'addition et la soustraction d'une part, et la multiplication et la division d'autre part.

EXEMPLES

```
PRINT 5 MOD 3
```

Le résultat est 2.

```
PRINT 5.99 MOD 3.1
```

Le résultat est 0 parce que 5.99 est arrondi à 6 et 3.1 à 3.



L'instruction MODE permet d'initialiser l'écran dans le mode spécifié.

FORMAT

MODE n° de mode

- Le **numéro de mode** peut prendre les valeurs 0, 1 ou 2.

COMMENTAIRES

Après exécution de l'instruction MODE, l'écran est effacé (dans la couleur de l'encre n° 0) et les curseurs graphique et texte sont placés à leur position d'origine. Les définitions des fenêtres (instruction WINDOW) sont d'autre part perdues.

Les caractéristiques de chaque mode sont énumérées dans les tableaux suivants :

Mode	Nb caractères par ligne	Nb lignes par page	Nb couleurs utilisables simultanément
0	20	25	16
1	40	25	4
2	80	25	2

Mode	Résolution graphique	Taille d'un pixel graphique
0	160 * 200	* * * * * * * * * * 4 * 2
1	320 * 200	* * * * 2 * 2
2	640 * 200	* * 1 * 2

Les instructions MOVE et MOVER placent le curseur graphique à un endroit spécifié de l'écran. L'instruction MOVE est associée à des coordonnées absolues tandis que l'instruction MOVER est associée à des coordonnées relatives.

FORMATS

MOVE coordonnée x, coordonnée y < , < n° d'encre >
< ,mode d'encre >

MOVER déplacement x, déplacement y < , < n° d'encre >
< ,mode d'encre >

- Le **numéro d'encre** est un paramètre qui n'existe pas sur le CPC 464. Il peut prendre des valeurs comprises entre 0 et 15 et spécifie la couleur de l'encre choisie.
- Le **mode d'encre** est un paramètre qui n'existe pas sur le CPC 464. Sur les autres modèles, il peut prendre les valeurs 0, 1, 2, 3 ou 4 suivant le mode d'interaction avec les couleurs de l'écran graphique.

COMMENTAIRES

Sur les modèles CPC 664 et 6128, il est possible de réaliser une opération logique entre le numéro de code de la couleur dans laquelle est affiché le point avant exécution de l'instruction MOVE ou MOVER, et le numéro de code de couleur correspondant au paramètre "n° d'encre" de cette instruction (voir les instructions DRAW et DRAWR pour de plus amples explications concernant le mécanisme de ces opérations logiques). Ces opérations sont les suivantes :

→

Valeur du paramètre	Opération
0	Normal
1	OU exclusif (XOR)
2	ET (AND)
3	OU (OR)

EXEMPLE DE PROGRAMME

L'exemple de programme des commandes DRAW et DRAWR utilise des instructions MOVE et MOVER pour positionner le curseur graphique avant chaque groupe d'instructions DRAW ou DRAWR.

Dans le programme illustrant les instructions SYMBOL et SYMBOL AFTER, la commande de la ligne 400 génère un déplacement du curseur graphique de 2 pixels vers la droite à chaque itération de la boucle FOR :

```
400 MOVE X2,Y1,2,1
```

Le troisième paramètre, 2, définit l'encre n° 2 pour le stylo graphique (c'est-à-dire la couleur turquoise vif, n° 20, qui a été initialisée ligne 180).

Le quatrième paramètre, 1, n'est utilisable que sur les modèles CPC 664 et 6128 ; il définit le mode d'encre correspondant à l'opérateur logique XOR (OU exclusif). L'ancienne couleur de chaque point tracé est combinée avec la nouvelle suivant l'opération spécifiée (XOR dans ce cas) ; cela permet de faire passer la voiture derrière les blocs jaunes et de la faire apparaître dans la couleur du fond lorsqu'elle chevauche les blocs turquoise.

L'initialisation du mode d'encre n'existe pas sur les modèles CPC 464 ; il est nécessaire d'ajouter la ligne suivante pour obtenir un effet similaire :

```
365 PRINT CHR$(23)+CHR$(X)
```

X pouvant avoir une valeur comprise entre 0 et 3, selon l'opération choisie.





La commande NEW efface de la mémoire centrale de l'ordinateur tous les programmes et toutes les données qui s'y trouvent.

FORMAT

NEW

COMMENTAIRES

Cette commande est généralement utilisée pour libérer la mémoire avant l'entrée d'un nouveau programme. Après son exécution, le système revient en mode direct mais l'écran n'est pas effacé. La commande NEW provoque la fermeture de tous les fichiers et positionne le mode Trace sur OFF (voir TRON/TROFF).

La réinitialisation complète du système peut être obtenue en appuyant simultanément sur les trois touches CTRL, SHIFT et ESC.

Les instructions ON...GOSUB et ON...GOTO provoquent le branchement du programme principal à un sous-programme (ON...GOSUB) ou à une autre partie du programme (ON...GOTO). La destination du branchement dépend de la valeur de l'expression spécifiée par ON et des numéros de lignes figurant après GOSUB ou GOTO.

FORMATS

ON expression numérique GOTO liste de numéros de lignes

ON expression numérique GOSUB liste de numéros de lignes

- L'**expression numérique** est automatiquement arrondie en valeur entière. Ce nombre doit être compris entre 0 et 255. Si ces limites ne sont pas respectées, le message d'erreur "Illegal Function Call", appel incorrect de fonction, est affiché.
- La **liste des numéros de lignes** correspond aux lignes d'instructions où doit s'effectuer le branchement, en fonction de la valeur de l'expression numérique. Les numéros de lignes doivent être séparés par des virgules, dans la liste.

RÈGLES DE SYNTAXE ET COMMENTAIRES

Lorsque l'expression numérique est égale à 1, le branchement est effectué au premier numéro de ligne de la liste écrite après GOTO ou GOSUB. Quand elle est égale à 2, le deuxième numéro de la liste est choisi pour le branchement (troisième numéro de la liste quand l'expression est égale à 3, etc.).

Lorsque l'expression numérique est égale à 0 ou lorsqu'elle est supérieure au nombre de numéros de lignes contenus dans la liste (tout en étant inférieure à 256), l'instruction ON...GOSUB ou ON...GOTO est ignorée et l'exécution normale du programme se poursuit.

Dans l'instruction ON...GOSUB, les numéros de lignes de la liste doivent correspondre au premier numéro de ligne d'un sous-

programme. Ceux-ci doivent se terminer par une instruction RETURN qui, lorsqu'elle est rencontrée, renvoie à l'exécution de l'instruction suivant ON...GOSUB dans le programme principal.

EXEMPLE DE PROGRAMME

Le programme ci-dessous affiche à l'écran quelques formules géométriques usuelles.

Selon la réponse fournie à la question posée à la ligne 10, la variable W prend la valeur 1,2 ou 3.

Il en résulte qu'à la ligne 20, le programme est dirigé vers les sous-programmes commençant aux lignes 100, 200 ou 300, selon le cas. Chaque sous-programme (ici très simple) se termine par une instruction RETURN, qui renvoie donc à la ligne 25.

A la ligne 40, le programme est redirigé soit vers la ligne 10, soit vers la ligne 50 (fin du programme).

```
10 INPUT "VOULEZ-VOUS LA FORMULE DONNANT
 LE PERIMETRE D'UN CERCLE (1), SA SURFAC
E (2) OU  LE VOLUME D'UNE SPHERE (3)";W
15 CLS
20 ON W GOSUB 100,200,300
25 PRINT
30 INPUT "UNE AUTRE FORMULE (OUI=1,NON=2
)";Z
35 PRINT
40 ON Z GOTO 10,50
50 END
100 PRINT "PERIMETRE D'UN CERCLE DE RAYO
N R :"
110 PRINT "2 x 3,14 x R"
120 RETURN
200 PRINT "SURFACE D'UN CERCLE DE RAYON
R :"
210 PRINT "3,14 x R X R"
220 RETURN
300 PRINT "VOLUME D'UNE SPHERE DE RAYON
R :"
310 PRINT "3,14 x R x R x R x 4/3"
320 RETURN
```

ON BREAK CONT, ON BREAK GOSUB et ON BREAK STOP

L'instruction ON BREAK CONT, qui n'existe que sur les modèles CPC 664 et 6128, ignore les interruptions générées par la double frappe de la touche ESC et permet au programme de se dérouler normalement.

L'instruction ON BREAK GOSUB entraîne le déroutement du programme vers un sous-programme en cas de double frappe de la touche ESC par l'utilisateur.

L'instruction ON BREAK STOP annule dans un programme le résultat des deux instructions précédentes.

FORMATS

ON BREAK CONT

ON BREAK GOSUB numéro de ligne

ON BREAK STOP

COMMENTAIRES

Il est possible, à tout moment, d'interrompre au clavier l'exécution d'un programme en appuyant deux fois sur la touche ESC. Si aucune instruction ON BREAK CONT (disponible uniquement sur les modèles CPC 664 et 6128) ou ON BREAK GOSUB n'existe dans le programme, l'ordinateur revient alors en mode direct. Dans le cas contraire, deux situations peuvent se présenter :

- L'instruction ON BREAK CONT désactive l'effet de la touche ESC.
- L'instruction ON BREAK GOSUB entraîne le branchement spécifié par le numéro de ligne suivant GOSUB.

EXEMPLE DE PROGRAMME

Les cinq lignes de programme suivantes complètent le programme de jeu donné en illustration de la rubrique "Opérateurs logiques". Dans sa version originale, le programme génère une suite de nombres aléatoires (ou plus exactement pseudo-aléatoires) grâce à la fonction RND(1). La suite de nombres est donc toujours la même, chaque fois que le programme est réinitialisé. L'amélioration proposée ici permet d'initialiser la liste d'une manière qui ne dépende pas d'un choix objectif du joueur.

```
  5 CLS
 10 PRINT "Taper 2 fois ESC pour commencer à jouer"
 15 ON BREAK GOSUB 500
 55 IF Z=1 THEN 60 ELSE 50
500 Z=1 : RETURN
```

Ces lignes peuvent soit être ajoutées directement au programme précité, soit être sauvegardées indépendamment sous un nom de programme spécifique qu'il suffira de fusionner avec le programme principal, au moment de l'exécution (la procédure est expliquée en illustration de la commande MERGE).

Ces quelques lignes de programme permettent de faire défiler la liste de nombres pseudo-aléatoires, la carte finalement choisie étant fonction du délai entre le moment où le message de la ligne 10 est affiché à l'écran et celui où le joueur frappe effectivement la touche ESC deux fois. Il suffit d'un écart de quelques fractions de seconde pour que la carte choisie soit différente. Une procédure semblable pourrait être utilisée avec la fonction TIME.

Tant que le joueur n'a pas pressé deux fois la touche ESC, de nouveaux nombres X1 et Y1 sont générés (car Z=0, valeur par défaut, la ligne 55 renvoyant dans ce cas l'exécution à la ligne 50). Lorsque le joueur génère une interruption, le programme est dérouté vers la ligne 500 où la valeur 1 est simplement donnée à la variable Z. Au retour du sous-programme, lorsque la ligne 55 est de nouveau exécutée, Z a cette fois la valeur 1, de sorte que le déroulement normal du programme à partir de la ligne 60 peut maintenant commencer.

Les instructions OPENIN et OPENOUT ouvrent respectivement le fichier spécifié en lecture ou en écriture.

FORMATS

OPENIN nom de fichier

OPENOUT nom de fichier

- Le **nom de fichier** est un paramètre caractérisant le fichier (huit caractères au maximum, voir la rubrique "Commandes AMSDOS").

COMMENTAIRES

Le terme *fichier* désigne indifféremment un programme ou un ensemble de valeurs (fichier de données). En BASIC Amstrad, il n'existe qu'un type de fichier, le type séquentiel. L'instruction OPENIN n'est utilisable que pour ouvrir un fichier ASCII (et non un fichier binaire).

Un fichier séquentiel stocke les informations les unes à la suite des autres, au fur et à mesure qu'elles lui sont adressées. Les données nouvelles sont ajoutées à la suite de celles qui existent déjà. Il n'est possible ni de modifier une donnée intermédiaire du fichier, ni d'accéder directement à celle-ci sans parcourir l'ensemble du fichier. Il existe un repère de fin de fichier (CRTL Z) de code ASCII 26 (1A en hexadécimal) qui, lorsqu'il est écrit, interdit l'accès à toute donnée du fichier qui se trouverait après ce repère.

Les données ne sont pas transférées une par une du fichier vers l'unité périphérique ou vice versa. Le transfert s'effectue par blocs, un bloc représentant le contenu de la zone de la mémoire centrale spécialement réservée au fichier (mémoire tampon). Ce n'est que lorsque cette mémoire tampon est remplie que s'effectue le transfert (ou lorsque le fichier est fermé au moyen d'une instruction CLOSEOUT, dans le cas d'un accès en mode écriture).

EXEMPLE

Le programme ci-après illustre les instructions relatives aux manipulations de fichier : il crée un fichier d'adresses et donne à l'utilisateur la possibilité de le modifier ou de consulter les fiches.

Les lignes 10 à 240 initialisent l'écran et proposent un choix dans un menu :

- lecture d'une fiche,
- initialisation de fiche,
- fin du traitement.

Les deux premières propositions entraînent la lecture du fichier sur cassette ou sur disquette (lignes 280 à 340). La ligne 280 ouvre le fichier "AGENDA" en lecture puis, tant que la fin du fichier n'est pas atteinte, chaque ligne de données lue est assignée à la variable T$(INC%). L'apparition du caractère de fin de fichier, CHR$(26), interrompt la boucle WHILE...WEND et la ligne 340 referme le fichier. Lors de la première exécution du programme, le fichier "AGENDA" n'existe pas ; pour éviter qu'une erreur de lecture se produise, il faut exécuter un petit programme qui crée le fichier "AGENDA" :

```
10 OPENOUT "AGENDA"
20 PRINT #9, CHR$(26);
30 CLOSEOUT
```

Ce programme ne devra plus être exécuté par la suite sous peine d'effacer irrémédiablement le fichier de données.

Le premier choix du menu propose à l'utilisateur d'entrer un nom puis, s'il correspond à une fiche, les données correspondantes sont affichées.

Le second choix permet d'entrer des noms et des adresses sur des fiches ; les données de chaque fiche sont concaténées dans une chaîne de caractères T$(INC%). Lorsque la saisie est terminée (lignes 1260 à 1300), toutes les fiches de l'agenda qui étaient stockées dans le tableau T$() sont écrites dans le fichier (lignes 1310 à 1350) ; puis le programme revient au menu.

```
10 BORDER 4
20 INK 0,1: INK 1,24: INK 2,0: INK 3,16
60 PAPER 0:PEN 1
80 DIM T$(100)
```

→

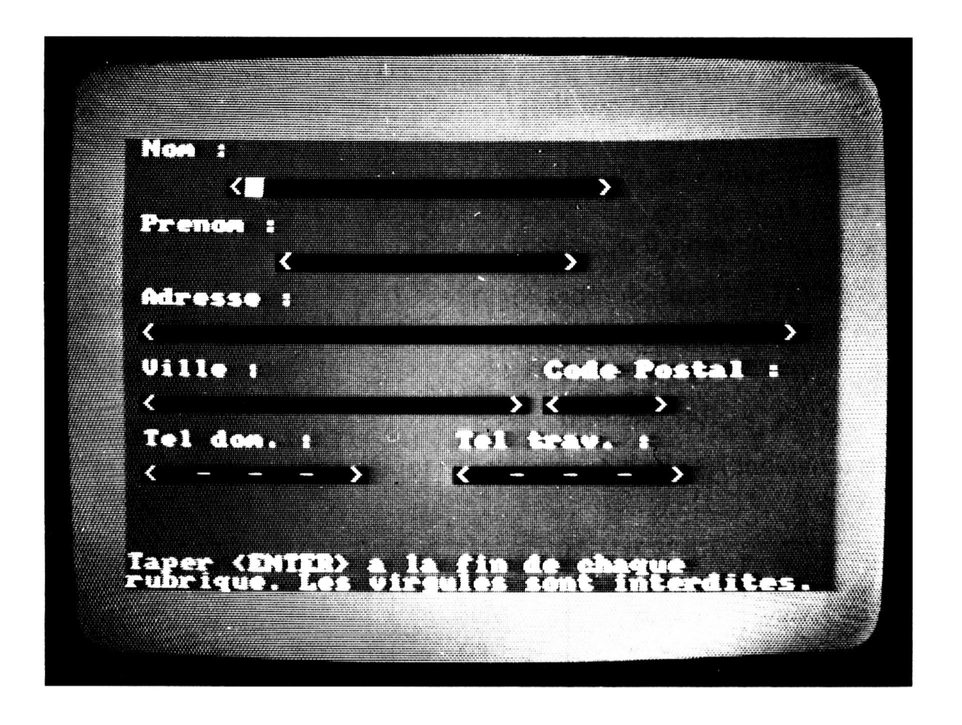
```
90 BL$=SPACE$(35)
100 CLS
110 LOCATE 8,3
120 PRINT "< FICHIER D'ADRESSES >"
130 LOCATE 2,10
140 PRINT "Lecture d'une fiche       (1)"
150 LOCATE 2,12
160 PRINT "Initialisation de fiches (2)"
170 LOCATE 2,14
180 PRINT "Fin de traitement        (3)"
190 LOCATE 2,21
200 PRINT "Taper le nombre correspondant"
210 LOCATE 2,22
220 PRINT "au traitement desire ";
230 C$=""
240 WHILE C$="" : C$=INKEY$ : WEND
250 C=VAL(C$)
260 IF C>3 OR C<1 THEN 230
270 IF C=3 THEN 370
280 OPENIN "AGENDA"
290 INC%=0
300 WHILE NOT EOF
310 INC%=INC%+1
320 LINE INPUT #9,T$(INC%)
330 WEND
340 CLOSEIN
350 ON C GOSUB 410,510
360 GOTO 100
370 CLS
380 LOCATE 10,12
390 PRINT "Acces au fichier termine !"
400 END
410 ' Lecture d'une fiche
420 CLS
430 LOCATE 3,5
440 PRINT "Nom de la personne recherchee :"
450 LOCATE 3,7 : INPUT NO$
460 FOR I%=1 TO INC%
470 NOM$=LEFT$(T$(I%),LEN(NO$))
480 IF NOM$=NO$ THEN GOSUB 1430
490 NEXT I%
500 RETURN
508 '
509 '*********************************
510 ' Initialisation d'une fiche
511 '*********************************
520 INC%=INC%+1
530 CLS
540 LOCATE 2,1 : PRINT "Nom :"
560 LOCATE 2,5 : PRINT "Prenom :"
```

```
580 LOCATE 2,9 : PRINT "Adresse :"
600 LOCATE 2,13 : PRINT "Ville :"
620 LOCATE 25,13 : PRINT "Code Postal :"
640 LOCATE 2,17 : PRINT "Tel dom. :"
660 LOCATE 20,17 : PRINT "Tel trav. :"
680 LOCATE 1,24
690 PRINT "Taper <ENTER> a la fin de chaque"
700 PRINT "rubrique. Les virgules sont interdites."
710 PAPER 2
720 PEN 3
740 LOCATE 7,3:PRINT "<";SPACE$(20);">"
750 LOCATE 10,7
760 PRINT "<";SPACE$(15);">"
770 LOCATE 2,11
780 PRINT "<";SPACE$(35);">"
790 LOCATE 2,15
800 PRINT "<";SPACE$(20);">"
810 LOCATE 25,15
820 PRINT "<";SPACE$(5);">"
830 LOCATE 2,19
840 PRINT "<  -  -  -  >"
850 LOCATE 20,19
860 PRINT "<  -  -  -  >"
870 LOCATE 8,3
880 T$(INC%)=""
890 INPUT "",NOM$
900 CH$=NOM$ : LONG%=20
920 GOSUB 1380
930 LOCATE 11,7
940 INPUT "",PREN$
950 CH$=PREN$ : LONG%=15
970 GOSUB 1380
980 LOCATE 3,11
990 INPUT "",ADR$
1000 CH$=ADR$ : LONG%=35
1020 GOSUB 1380
1030 LOCATE 3,15
1040 INPUT "",VIL$
1050 CH$=VIL$ : LONG%=20
1070 GOSUB 1380
1080 LOCATE 26,15
1090 INPUT "",CODE$
1100 CH$=CODE$ : LONG%=5
1120 GOSUB 1380
1130 LOCATE 3,19
1140 INPUT "",TELD$
1150 CH$=TELD$ : LONG%=11
1170 GOSUB 1380
1180  LOCATE 21,19
1190 INPUT "",TELT$
```

→

```
1200 CH$=TELT$ : LONG%=11
1220 GOSUB 1380
1230 PAPER 0 : PEN 1
1250 CLS
1260 PRINT "Une autre (O/N) ?"
1270 C$=""
1280 WHILE C$="" : C$=INKEY$ : WEND
1290 IF C$<>"O" AND C$<>"N" THEN 1270
1300 IF C$="O" THEN 510
1310 OPENOUT "AGENDA"
1320 FOR J%=1 TO INC%
1330 PRINT #9,T$(J%)
1340 NEXT J%
1350 CLOSEOUT
1360 CLS
1370 RETURN
1378 '
1379 '**********************************
1380 ' Ecriture d'une fiche dans une chaine
1381 '**********************************
1390 CH$=CH$+BL$
1400 CH$=LEFT$(CH$,LONG%)
1410 T$(INC%)=T$(INC%)+CH$
1420 RETURN
1428 '
1429 '**********************************
1430 ' Affichage d'une fiche
1431 '**********************************
1440 CLS
1450 PAPER 2 : PEN 3
1470 LOCATE 8,3
1480 PRINT "<   FICHE -->          >"
1490 LOCATE 21,3
1500 PRINT I%;
1510 LOCATE 2,8
1520 PRINT LEFT$(T$(I%),20)
1530 LOCATE 2,10
1540 PRINT MID$(T$(I%),21,15)
1550 LOCATE 2,12
1560 PRINT MID$(T$(I%),36,35)
1570 LOCATE 2,14
1580 PRINT MID$(T$(I%),91,5)
1590 LOCATE 17,14
1600 PRINT MID$(T$(I%),71,20)
1610 LOCATE 2,16
1620 PRINT "Tel dom.  ";MID$(T$(I%),96,11)
1630 LOCATE 2,18
1640 PRINT "Tel trav. ";RIGHT$(T$(I%),11)
1650 PAPER 0
1660 PEN 1
```

```
1670 LOCATE 1,25
1680 PRINT "Taper une touche pour continuer ";
1690 C$=""
1700 WHILE C$="" : C$=INKEY$ : WEND
1710 RETURN
```



Ces opérateurs effectuent des opérations logiques (ou booléennes) sur des valeurs numériques. Ils sont généralement utilisés dans un programme pour coupler deux relations (ou plus) en renvoyant une valeur vraie ou fausse qui détermine la suite de l'exécution du programme. Le résultat de l'opération est un nombre qui est *faux* s'il est égal à 0 ou *vrai* s'il est égal à une autre valeur.

Les opérateurs logiques sont les suivants, par ordre de préséance lors de l'exécution (l'opérateur NOT n'existe pas sur le CPC 464) :

NOT (complément logique)
AND (conjonction)
OR (disjonction)
XOR (ou exclusif)

Ils renvoient les résultats suivants (en fonction des valeurs de X et Y) :

X	Y	NOT X	X AND Y	X OR Y	X XOR Y
Vrai	Vrai	Faux	Vrai	Vrai	Faux
Vrai	Faux	Faux	Faux	Vrai	Vrai
Faux	Vrai	Vrai	Faux	Vrai	Vrai
Faux	Faux	Vrai	Faux	Faux	Faux

Par exemple (3e ligne), lorsque X est faux, NOT X est vrai. Y étant vrai, une seule des deux valeurs X et Y est vraie, donc les opérations X OR Y et X XOR Y sont toutes deux vraies. Comme X et Y ne sont pas simultanément vrais, le résultat de l'opération X AND Y est faux. Dans l'instruction IF...THEN...ELSE suivante :

```
60 IF A < 50 AND A > 10 THEN PRINT "10 < A < 50" ELSE 100
```

le résultat de l'expression IF est vrai si A est compris entre 10 et 50. Dans ce cas, la clause THEN est exécutée. Dans le cas contraire, la clause ELSE renvoie le programme à la ligne 100.

Les opérations logiques sont effectuées après les tests relationnels (<, >, =, etc.) et les opérations numériques. L'ordre de préséance

des opérations peut toutefois être modifié en utilisant des parenthèses. Les opérations figurant à l'intérieur de parenthèses sont toujours effectuées les premières, l'ordre de préséance habituel demeurant respecté à l'intérieur de ces parenthèses.

Afin de se familiariser avec les opérateurs logiques, on pourra vérifier que :

```
IF A THEN X=(A AND B) ELSE X=(A OR B)
```

est équivalent à X=B.

```
IF A THEN X=(A OR B) ELSE X=(A AND B)
```

est équivalent à X=A.

```
IF A AND B THEN X=A ELSE X=B
```

est équivalent à X=B.

```
IF A OR B THEN X=A ELSE X=B
```

est équivalent à X=A.

```
IF A THEN X=B ELSE X=(A OR B)
```

est équivalent à X=B.

```
IF A AND B THEN X=A ELSE X=(A OR B)
```

est équivalent à X=A OR B.

EXEMPLE DE PROGRAMME

Le programme de jeu donné ci-après consiste à deviner la carte choisie au hasard par l'ordinateur, dans un jeu de 52 cartes. Après chaque proposition du joueur, l'ordinateur indique si la couleur proposée est la bonne et si la valeur de la carte choisie est supérieure, inférieure ou égale à celle qu'il faut deviner. A chaque nouvel essai, l'ordinateur rappelle le résultat des coups précédemment joués. Le joueur a droit à trois essais. Il dispose d'un essai supplémentaire lorsqu'il a la chance de voir s'afficher le JOKER après un troisième essai infructueux.

```
10 CLS
20 A$(1)="PIQUE   " : A$(2)="COEUR " : A$
(3)="CARREAU" : A$(4)="TREFLE"
30 P1$="BONNE COULEUR" : P2$="MAUVAISE C
OULEUR"
40 Q1$="VALEUR TROP FAIBLE" : Q2$="VALEU
R TROP FORTE" : Q3$="VALEUR EXACTE"
50 X1=1+INT(RND(1)*4) : Y1=1+INT(RND(1)*
13)
58 '
59 '*********************************
60 ' Proposition d'une carte par le joue
ur
61 '*********************************
70 N=1
80 PRINT : PRINT "ESSAI NUMERO";N :PRINT

90 INPUT "Valeur de la carte (de 1 a 13,
 valet=11,dame=12, roi=13) ";Y(N)
95 PRINT
100 INPUT "Couleur choisie (PIQUE=1, COE
UR=2,        CARREAU=3, TREFLE=4) ";X(N)
110 IF X(N)=10 THEN END
118 '
119 '********************************
120 ' Test de comparaison
121 '********************************
130 PRINT : X=X(N) : Y=Y(N)
140 IF X=X1 XOR Y=Y1 THEN IF X=X1 AND Y<
Y1 THEN P$=P1$:Q$=Q1$ ELSE IF X=X1 AND Y
>Y1 THEN P$=P1$:Q$=Q2$ ELSE P$=P2$:Q$=Q3
$ ELSE IF X=X1 AND Y=Y1 THEN PRINT "BRAV
O VOUS AVEZ GAGNE " : GOTO 220 ELSE IF Y
<Y1 THEN P$=P2$:Q$=Q1$ ELSE P$=P2$:Q$=Q2
$
150 P$(N)=P$ : Q$(N)=Q$
158 '
159 '********************************
160 ' Affichage des resultats
161 '********************************
165 CLS
170 PRINT : F=INT(RND(1)*2) : PRINT "Res
ultat des essais effectues" : PRINT
180 FOR J=1 TO N
190 PRINT Y(J);" de ";A$(X(J));TAB(17)"-
-> ";P$(J) : PRINT TAB(21)Q$(J) : PRINT
200 NEXT J : N=N+1 : IF N<4 THEN 80 ELSE
 IF F=0 AND N<5 THEN PRINT "JOKER : essa
i supplementaire" : GOTO 80
210 PRINT "Pas de chance ! Vous avez per
```

```
du ; l'ordinateur avait choisi le ";Y1;"
 de ";A$(X1)
220 PRINT : PRINT "L'ordinateur a choisi
 un nouveau numero"
230 PRINT "Pour interrompre le jeu, tape
r 10 pour  la couleur"
240 GOTO 50
```

Le cœur de ce programme (ligne 140) est constitué de cinq instructions IF...THEN...ELSE imbriquées utilisant les opérateurs logiques XOR et AND. Elles permettent à l'ordinateur de tester la proposition et de choisir ses réponses.

Les lignes 20 à 40 affectent des valeurs de chaînes aux variables A$ à Q3$; cette procédure d'assignation permet d'éviter la répétition de la même chaîne de caractères à plusieurs reprises, dans le programme.

La ligne 50 génère au hasard deux nombres entiers, X1 (compris entre 1 et 4) et X2 (compris entre 1 et 13). X1 correspond à la couleur et X2 à la valeur de la carte à deviner (en respectant les conventions des lignes 100 et 90).

N est le compteur du nombre de tentatives effectuées par le joueur. Il est initialisé à 1 au début du programme (ligne 70) puis est incrémenté d'une unité avant chaque nouvel essai (ligne 200).

La valeur et la couleur de la carte proposée par le joueur en réponse aux instructions INPUT des lignes 90 et 100 sont respectivement stockées dans les variables indicées Y(N) et X(N). L'utilisation de variables indicées a pour seul but de conserver en mémoire, pour ensuite les afficher, les cartes choisies lors des (N – 1) essais précédents. La ligne 110 teste la condition de sortie du programme.

La ligne 140 détermine si la valeur et la couleur de la $N^{ième}$ carte proposée par le joueur correspondent l'une et/ou l'autre à celles qu'a choisies l'ordinateur. L'appariement des expressions IF et des clauses THEN et ELSE est le suivant (voir l'instruction correspondante) :

- Le premier IF et le premier THEN sont appariés au troisième ELSE.
- Le deuxième IF et le deuxième THEN sont appariés au premier ELSE.
- Le troisième IF et le troisième THEN sont appariés au deuxième ELSE.
- Le quatrième IF et le quatrième THEN sont appariés au quatrième ELSE.

• Le cinquième IF et le cinquième THEN sont appariés au cinquième ELSE.

En tenant compte de cette règle d'appariement et du tableau de vérité des opérateurs logiques donné plus haut, la ligne 140 est structurée de la manière suivante :

```
IF X=X1 XOR Y=Y1
         THEN IF X=X1 AND Y < Y1
                  THEN "couleur correcte et valeur trop petite"
                  ELSE IF X=X1 AND Y > Y1
                           THEN "couleur correcte et valeur trop grande"
                           ELSE "couleur fausse et valeur exacte"
         ELSE IF X=X1 AND Y=Y1
                  THEN "gagné !!!"
                  ELSE IF Y > Y1
                           THEN "couleur fausse et valeur trop grande"
                           ELSE "couleur fausse et valeur trop petite"
```

Les six possibilités étant ainsi différenciées, la ligne 150 permet d'assigner les valeurs exactes aux messages à afficher sur l'écran. Pour des raisons pratiques, on utilise les variables alphanumériques P$ et Q$, car les noms des variables P$(N) et Q$(N) occupent trop de place et, s'ils étaient utilisés à la ligne 140, celle-ci dépasserait 255 caractères et deviendrait trop longue.

La dernière partie du programme (lignes 160 à 240) concerne l'affichage à l'écran des résultats et les redirections du programme en fonction du nombre de coups déjà joués. Un groupe d'instructions permet de donner une quatrième chance au joueur : un nombre entier aléatoire 0, 1 ou 2 est généré et stocké dans la variable F (ligne 170). Lorsque les conditions $N < 5$ (qui est équivalente, d'après le test conditionnel qui précède, à $N = 4$) et $F = 0$ sont simultanément vérifiées, un essai supplémentaire est accordé. Dans le cas contraire, la carte qu'avait choisie l'ordinateur est révélée et le jeu recommence avec tirage d'une nouvelle carte (renvoi à la ligne 50).

Quelques instructions peuvent encore être ajoutées à ce programme afin d'initialiser de manière différente le générateur de nombres aléatoires chaque fois que le programme est lancé. Ces instructions sont données en illustration de la rubrique ON BREAK... Elles mettent en jeu la commande ON BREAK CONT (disponible uniquement sur les

modèles CPC 664 et 6128) et la touche Esc ne peut donc plus être utilisée pour sortir du programme. C'est la raison pour laquelle une procédure de sortie particulière a été adoptée ici (lignes 110 et 230).

D'autre part, quatre lignes de programme supplémentaires, décrites en regard de la rubrique ''Gestion des erreurs'', permettent de s'affranchir d'une entrée clavier erronée en réponse aux questions posées.

```
Resultat des essais effectues
 3  de TREFLE     --> MAUVAISE COULEUR
                      VALEUR TROP FAIBLE
 7  de COEUR      --> MAUVAISE COULEUR
                      VALEUR TROP FAIBLE
 11  de PIQUE     --> BONNE COULEUR
                      VALEUR TROP FAIBLE
 12  de PIQUE     --> BONNE COULEUR
                      VALEUR TROP FAIBLE
Pas de chance ! Vous avez perdu : l'ordi
nateur avait choisi le  13  de PIQUE
L'ordinateur a choisi un nouveau numero
Pour interrompre le jeu, taper 10 pour
la couleur
```





L'instruction ORIGIN redéfinit les coordonnées de l'origine de l'écran graphique et permet de définir une nouvelle fenêtre graphique.

FORMAT

ORIGIN coord x, coord y < , gauche, droite, haut, bas >

- **coord x** et **coord y** sont les coordonnées graphiques de la nouvelle origine.
- **gauche, droite, haut** et **bas** sont les coordonnées de la nouvelle fenêtre graphique.

COMMENTAIRES

Après exécution de la commande ORIGIN, le curseur graphique est placé à la nouvelle origine ; tout se passe comme si une instruction MOVE 0,0 avait été exécutée.

EXEMPLE DE PROGRAMME

Le programme illustrant la commande PLOT utilise l'instruction ORIGIN pour définir une nouvelle origine. Les lignes 280 à 330 tracent un cadre en conservant l'origine par défaut (en bas à gauche de l'écran), puis une nouvelle origine est définie au centre de l'écran :

```
350 ORIGIN 319,199
```

Cette instruction est destinée à placer la figure au centre de l'écran sans qu'il soit nécessaire de faire une translation des coordonnées de chaque point.

L'instruction PAPER initialise la couleur du fond pour la totalité de l'écran texte ou pour une fenêtre particulière.

FORMAT

PAPER < # numéro de canal, > numéro d'encre

- **numéro de canal** est un nombre compris entre 0 et 7 qui caractérise une fenêtre particulière.
- **numéro d'encre** est un nombre compris entre 0 et 15.

COMMENTAIRES

Lorsque le numéro de canal n'est pas précisé, la valeur 0 est prise par défaut.

Sur les modèles CPC 664 et 6128, deux cas peuvent se présenter lorsqu'un caractère est écrit sur l'écran :

- Si le stylo est initialisé en mode opaque (PEN #n1,n2,0), la matrice réservée au caractère (32*16 pixels en mode 0, 16*16 pixels en mode 1 et 8*16 pixels en mode 2) est remplie par la couleur de l'encre correspondant au papier.
- Si le stylo est initialisé en mode transparent (PEN #n1,n2,1), la matrice réservée au caractère reste dans la couleur du fond.

Après l'exécution d'une commande CLS, l'écran est initialisé dans la couleur du papier.





PEEK lit la valeur de l'octet, en mémoire centrale de l'ordinateur (mémoire RAM), dont l'adresse est donnée comme argument à la fonction.

FORMAT

PEEK(n)

- **n** est un nombre entier compris entre 0 et 65535 (des valeurs négatives peuvent être données comme argument, l'ordinateur prenant alors le complément à 2, c'est-à-dire qu'il leur ajoute 65535).

COMMENTAIRES

Cette fonction permet une lecture directe du contenu de la mémoire vive RAM de l'ordinateur. Elle est complémentaire de l'instruction POKE, qui permet d'écrire directement des données à une adresse spécifiée de la mémoire.

Supposons qu'un nombre ait été stocké à une adresse D de la mémoire RAM et que l'on veuille lire sa valeur au moyen de l'instruction PEEK. Pour comprendre la procédure à suivre, il est nécessaire de revenir sur le mode de stockage des informations par l'ordinateur.

La mémoire est organisée en octets, chaque octet représentant l'association de 8 bits qui ne peuvent prendre chacun que la valeur 0 ou 1. Supposons qu'un octet contienne la valeur 100. Cette valeur peut s'écrire 01100010 en binaire (voir les fonctions BIN$ et HEX$), c'est-à-dire que les bits de cet octet auront les valeurs respectives (de droite à gauche) 0,1,0,0,0,1,1,0.

Une méthode commode de représenter cela est de donner un rang à chacun des bits et de lui affecter un poids dépendant de son rang. Ce poids est égal à 2 élevé à la puissance du rang, ce qui donne pour un octet :

Rang du bit	7	6	5	4	3	2	1	0
Poids du bit	128	64	32	16	8	4	2	1

La valeur de l'octet est simplement obtenue en additionnant le poids des bits qui sont positionnés (c'est-à-dire ceux qui ont la valeur 1) et en ignorant les bits qui sont à 0. (On peut effectivement vérifier que 100=2+32+64, ce qui correspond au positionnement des bits de rang 1, 5 et 6.) Si tous les bits sont positionnés, l'octet aura la valeur 255, ce qui explique pourquoi un nombre ayant une valeur supérieure doit être stocké sur deux octets.

Les bits du second octet, dit *octet de poids fort*, ont un rang allant de 8 à 15, soit, en fonction des règles précédemment énoncées, un poids allant de 256 à 32768 (2^{15}). En considérant ce mode de stockage sur deux octets, il est facile de montrer qu'il est possible de décomposer un nombre en deux facteurs résultant de sa division par 256 :

- le quotient entier de la division est stocké dans l'octet de poids fort (c'est celui qui aura l'adresse la plus élevée) ;
- le reste de la division est stocké dans l'octet de poids faible.

Pour avoir le nombre stocké à l'adresse D, il est donc nécessaire d'additionner le contenu pondéré des octets d'adresses D et D+1, c'est-à-dire d'écrire l'instruction :

```
PEEK(D) + 256 * PEEK(D+1)
```

L'argument de l'instruction PEEK peut également être exprimé en notation hexadécimale ; il doit dans ce cas être précédé du préfixe &H. Par exemple :

```
PEEK(&HF00)
```

retourne la valeur de l'octet d'adresse 3840 (en décimal).

Des explications complémentaires concernant l'adressage direct de la mémoire sont données en regard de la fonction @.



L'instruction PEN initialise la couleur de l'encre à utiliser, pour la fenêtre indiquée.

FORMAT

PEN < # numéro de canal, > < numéro d'encre > < ,mode du fond >

- **numéro de canal** est un nombre compris entre 0 et 7 qui caractérise une fenêtre particulière.
- **numéro d'encre** est un nombre compris entre 0 et 15.
- **mode du fond** est un paramètre qui n'existe pas sur le CPC 464 ; il peut prendre la valeur 0 ou 1. 0 correspond à un fond opaque, tandis que 1 correspond à un fond transparent.

COMMENTAIRES

Lorsque le numéro de canal n'est pas précisé, la valeur 0 est prise par défaut.

Le programme suivant, destiné aux modèles CPC 664 et 6128, affiche un message en mode transparent et un autre en mode opaque :

```
10 PAPER 0
20 CLS
30 INK 2,7
40 PAPER 2
50 PEN 1,0
60 LOCATE 5,10
70 PRINT "MESSAGE EN MODE OPAQUE"
80 PEN 1,1
90 LOCATE 5,15
100 PRINT "MESSAGE EN MODE TRANSPARENT"
```

La constante PI est égale à 3,14159265 (valeur arrondie).

FORMAT

PI

COMMENTAIRES ET EXEMPLE DE PROGRAMME

Cette constante est essentiellement utilisée dans les calculs trigonométriques. Contrairement à beaucoup d'autres BASIC, celui de l'Amstrad ne possède pas de fonction CIRCLE permettant de tracer directement un cercle. Le programme suivant permet d'effectuer ce tracé à l'aide des fonctions SIN et COS et de la constante PI.

```
10 CLS
20 PRINT "ENTREZ (DANS L'ORDRE ET SEPARE
S PAR UNE"
30 PRINT "VIRGULE) LES COORDONNEES HORIZ
ONTALE"
40 INPUT "ET VERTICALE DU CENTRE DU CERC
LE";X,Y
50 PRINT
60 INPUT "RAYON DU CERCLE";R
70 MODE 2
80 ORIGIN 319,199
90 MOVE X,Y+R
100 FOR I=0 TO 2*PI STEP PI/200
110 DRAW X+R*SIN(I),Y+R*COS(I)
120 NEXT I
```

Les instructions PLOT et PLOTR dessinent un point graphique à un endroit spécifié de l'écran. L'instruction PLOT est associée à des coordonnées absolues, tandis que l'instruction PLOTR est associée à des coordonnées relatives.

FORMATS

PLOT coordonnée x, coordonnée y<, <n° d'encre>
<,mode d'encre>

PLOTR déplacement x, déplacement y<, <n° d'encre>
<,mode d'encre>

- Le **numéro d'encre** est un paramètre qui peut prendre des valeurs comprises entre 0 et 15 en fonction du numéro de l'encre choisie.
- Le **mode d'encre** est un paramètre qui n'existe pas sur le CPC 464 ; il peut prendre les valeurs 0, 1, 2, 3 ou 4 suivant le mode d'interaction avec les couleurs de l'écran graphique.

COMMENTAIRES

Sur les modèles CPC 664 et 6128, il est possible de réaliser une opération logique entre la couleur dans laquelle est affiché le point graphique avant exécution de l'instruction PLOT ou PLOTR et la couleur spécifiée par le paramètre "n° d'encre" (voir les instructions DRAW et DRAWR pour de plus amples informations concernant le mécanisme de ces opérations logiques). Ces opérations sont les suivantes :

Valeur du paramètre	Opération
0	Normal
1	OU exclusif (XOR)
2	ET (AND)
3	OU (OR)

EXEMPLE DE PROGRAMME

Le programme ci-après utilise l'instruction PLOT pour tracer sur l'écran des points qui représentent l'interférence de deux mouvements vibratoires de fréquences différentes ; les figures obtenues s'appellent *figures de Lissajous*. Les coordonnées des points sont les suivantes :

$$x = \sin(rap1 * T)$$

$$y = \sin(rap2 * T + D)$$

Lors de l'exécution du programme, l'utilisateur doit entrer une valeur D qui représente la différence de phase entre les deux coordonnées (lignes 110 à 180). Il doit ensuite indiquer (lignes 190 à 250) deux valeurs, RAP1 et RAP2, qui correspondent au rapport des deux angles.

L'ordinateur passe en mode graphique haute résolution puis dessine un cadre sur les bords de l'écran.

La ligne 350 permet de déplacer l'origine au centre de l'écran.

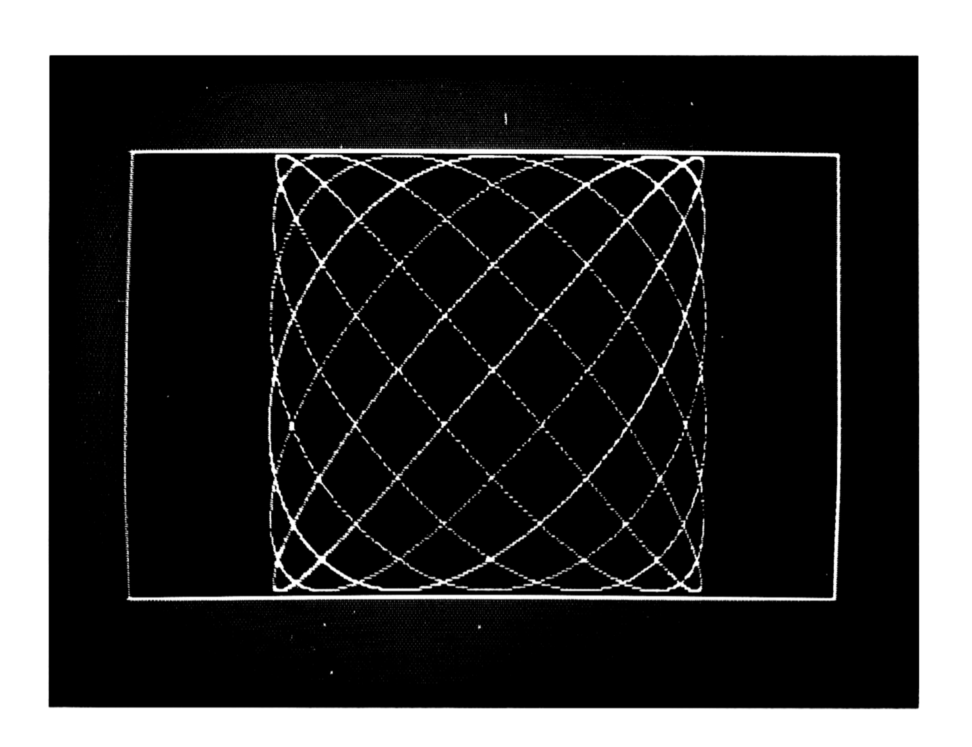
Ensuite, les lignes 340 à 390 tracent la fonction ; le curseur graphique est placé sur le premier point et 5 000 points sont ensuite tracés. On remarque que la valeur 195, qui est assignée à la variable R (ligne 60), représente pratiquement la moitié de l'axe vertical ; cela permet au dessin d'occuper la quasi-totalité de l'écran sans sortir des limites.

Une figure simplifiée aurait pu être générée en modifiant les lignes 370 et 380 :

```
370 FOR T=0 TO 2*PI STEP PI/200
380 DRAW SIN(RAP1*T)*R,SIN(RAP2*T+D)*R
```

```
10 CLS
20 MODE 1
30 BORDER 3
40 INK 0,0
50 INK 1,16
60 R=195
70 WHILE 1
80 CLS
90 LOCATE 10,5
100 PRINT "Figures de Lissajous"
110 LOCATE 1,10
120 PRINT "Dephasage (nombre"
```
→

```
130 PRINT "compris entre 0 et"
140 PRINT "2, il sera"
150 PRINT "multilpie par PI.)"
160 LOCATE 26,10
170 INPUT ": ",D
180 D=D*PI
190 LOCATE 1,16
200 PRINT "Rapports des angles :"
210 PRINT "(nombres entiers) "
220 LOCATE 10,19
230 INPUT "premiere valeur : ",RAP1
240 LOCATE 10,20
250 INPUT "seconde valeur  : ",RAP2
260 CLS
268 '
269 '**********************************
270 ' Trace du cadre
271 '**********************************
280 MODE 2
290 MOVE 0,0
300 DRAW 639,0
310 DRAW 639,399
320 DRAW 0,399
330 DRAW 0,0
338 '
339 '**********************************
340 ' Trace de la fonction
341 '**********************************
350 ORIGIN 319,199
360 MOVE 0,SIN(D)*R
370 FOR T=0 TO 2*PI STEP 2*PI/5000
380 PLOT SIN(RAP1*T)*R,SIN(RAP2*T+D)*R
390 NEXT T
400 LOCATE 2,24
410 PRINT "Fin!"
420 C$=""
430 WHILE C$="" : C$=INKEY$ : WEND
440 MODE 1
450 WEND
```





POKE

Instruction

POKE permet de stocker un octet de données à une adresse de la mémoire centrale (RAM) spécifiée.

FORMATS

POKE n,m

- **n** est l'adresse de la mémoire qui doit recevoir les données ; il s'agit d'un nombre entier compris entre 0 et 65535.
- **m** représente la valeur à stocker, sous forme d'un nombre entier compris entre 0 et 255.

COMMENTAIRES

Le format de stockage des données (m, nombre d'octets) est expliqué à propos de la fonction PEEK, qui est complémentaire de POKE. Cette instruction écrivant directement dans la mémoire centrale de l'ordinateur, toute erreur peut avoir des conséquences fâcheuses sur le fonctionnement de celui-ci. Cependant, elle ne peut altérer en aucune manière le matériel puisque, à la mise hors tension, le contenu de la mémoire vive est effacé.

Le programme donné en illustration de l'instruction MEMORY utilise l'instruction POKE pour écrire trois instructions en langage machine.

Les fonctions POS et VPOS renvoient respectivement, en mode texte, la position horizontale (numéro de colonne) et la position verticale (numéro de ligne) du curseur tel qu'il apparaît sur l'écran.

FORMATS

POS (# numéro de canal)

VPOS (# numéro de canal)

- **numéro de canal** peut prendre des valeurs comprises entre 0 et 9 pour la fonction POS ; il spécifie l'écran entier ou une fenêtre (0 à 7), l'imprimante (8), le magnétophone ou le lecteur de disque (9).
- **numéro de canal** peut prendre des valeurs comprises entre 0 et 7 pour la fonction VPOS.

COMMENTAIRES

La fonction POS(#8) renvoie la position de la tête d'impression, valeur qui est comprise entre 1 et 132 si l'imprimante n'a pas été initialisée par une commande WIDTH.

La fonction POS(#9) renvoie la position du pointeur dans la mémoire tampon, c'est-à-dire le nombre de caractères qui ont été envoyés depuis le dernier retour chariot, CHR$(&H0D). S'il ne contient aucun caractère, la valeur 1 est retournée.

Quel que soit le mode, la fonction VPOS renvoie une valeur comprise entre 1 et 25.

La fonction POS retourne une valeur dont les limites sont liées au mode en cours : entre 1 et 20 pour le mode 0, entre 1 et 40 pour le mode 1, entre 1 et 80 pour le mode 2.

Un exemple d'utilisation de la fonction POS est donné dans le programme illustrant l'instruction KEY.

Les fonctions XPOS et YPOS sont associées au curseur graphique et ont un rôle équivalent à celui des fonctions POS et VPOS. L'instruction LOCATE permet de positionner le curseur texte à l'écran (ou sur l'imprimante). CURSOR permet de rendre le curseur texte visible ou invisible.

L'instruction PRINT affiche à l'écran ou imprime les données ou les messages spécifiés. Une syntaxe particulière permet d'écrire les mêmes données dans un fichier stocké sur cassette ou sur disque.

FORMAT

PRINT < N° de canal, > < liste d'expressions > < ; >

- La **liste d'expressions** est une suite d'expressions numériques et/ou de chaînes de caractères. Des expressions consécutives sont séparées par des virgules, des espaces ou des points-virgules. Toute constante chaîne de caractères doit se trouver entre guillemets. Le mot clé PRINT peut être remplacé par un point d'interrogation.
- Le **N° de canal** est un nombre entier compris entre 0 et 9. La valeur par défaut est 0. Les valeurs comprises entre 0 et 7 correspondent aux fenêtres définies par l'instruction WINDOW. La valeur 8 correspond à une sortie sur imprimante. Le canal 9 est utilisé pour adresser l'unité standard de sortie (magnétocassette ou unité de disque), c'est-à-dire pour écrire les données dans un fichier ouvert.

RÈGLES DE SYNTAXE ET COMMENTAIRES

Lorsque l'instruction PRINT n'est pas suivie d'une liste d'expressions, une ligne blanche est affichée (ou imprimée). La ponctuation à l'intérieur de la liste d'expressions détermine le format de l'affichage.

Des expressions séparées par un point-virgule ou par un ou plusieurs espaces sont affichées les unes à la suite des autres. Par exemple, les deux lignes suivantes :

```
PRINT "Illustration";"de";"l'instruction";"PRINT"
PRINT "Illustration"   "de" "l'instruction"   "PRINT"
```

sont équivalentes (elles peuvent également être écrites en mode programme) et conduisent à l'affichage (sans espace entre les mots) de :

Si la liste d'expressions est trop longue pour que toutes puissent être écrites sur la même ligne, l'affichage se poursuit à la ligne suivante.

Lorsque les expressions de la liste sont séparées par une virgule, BASIC utilise la tabulation automatique dont la taille est définie par l'instruction ZONE : chaque ligne sur l'écran ou sur le papier est divisée en zones de Z caractères, Z étant la valeur du paramètre de ZONE. Chaque expression est affichée au début de la première zone non occupée, même partiellement, par l'expression précédente. Plusieurs virgules consécutives peuvent figurer dans une instruction PRINT. Dans ce cas, les virgules supplémentaires définissent autant de zones blanches.

Lorsqu'une liste d'expressions se termine par une virgule, un point-virgule ou une fonction SPC ou TAB, l'affichage commandé par l'instruction PRINT suivante s'effectue sur la même ligne, en respectant les espaces. Dans le cas contraire, l'affichage commence au début d'une nouvelle ligne.

Les nombres affichés à l'écran ou sur l'imprimante sont toujours suivis d'un espace. Ils sont également précédés d'un espace s'il s'agit de nombres positifs ou d'un signe moins (–) s'il s'agit de nombres négatifs.

L'instruction PRINT USING permet de modifier le format d'impression des nombres et des chaînes de caractères à l'écran et sur l'imprimante.

Les fonctions TAB et SPC peuvent être utilisées en relation avec l'instruction PRINT pour modifier les caractéristiques de l'affichage. L'instruction WIDTH détermine le nombre maximal de caractères imprimés sur une ligne.

L'instruction PRINT...USING permet d'envoyer des données formatées vers l'unité périphérique spécifiée.

FORMAT

PRINT < # N° de canal, > < liste d'expressions >
USING chaîne modèle < ,séparateur et expression >

- Les paramètres qui suivent le mot clé PRINT sont les mêmes que ceux qui sont décrits à propos de l'instruction PRINT.
- La **chaîne modèle** est une chaîne de caractères (variable ou constante) qui définit les caractéristiques de l'affichage.
- Le **séparateur** est une virgule ou un point-virgule.

COMMENTAIRES

Les différentes options possibles pour la "chaîne modèle" dépendent de la nature (numérique ou alphanumérique) des expressions à afficher ou à imprimer. Pour l'impression et l'affichage de nombres, les seuls caractères admis sont les suivants :

. , £ * $ + – ^

Leur signification et la manière dont ils peuvent être combinés dans la chaîne modèle sont expliquées à propos de l'instruction DEC$.

Pour formater une expression alphanumérique, les symboles suivants peuvent être utilisés dans la chaîne modèle :

! \ &

! Seul le premier caractère de l'expression sera affiché ou imprimé.

\ \	Les caractères \ indiquent une longueur d'impression égale au nombre d'espaces plus deux compris entre les deux symboles \. Si la chaîne est plus courte que le modèle, elle est complétée à droite par des blancs.
&	Spécifie que la chaîne entière doit être affichée, quels que soient sa longueur ou son format.

Exemple

```
10 A$="cor":B$="au":C$="pied"
20 PRINT USING "!";A$;B$;C$
30 PRINT USING "\ \";A$;B$
```

conduira à l'affichage suivant :

```
cap
corau
```

L'instruction DATA fournit des valeurs (numériques ou de chaînes) lues en cours d'exécution du programme par l'instruction READ.

FORMATS

DATA valeur < ,valeur > ...

READ variable < ,variable > ...

Exemples

```
DATA 3,9,15,16
DATA Louis,Charles,Henri,16,9,4
DATA "DUVAL,012 29 34","DUPONT,720 08 38"
READ C$,N,H1
DATA B$(1),B$(4)
```

RÈGLES DE SYNTAXE

Chaque instruction DATA et chaque instruction READ peuvent comporter autant de valeurs et de variables qu'une ligne de programme en contient (jusqu'à 255 caractères). Ces valeurs et ces variables doivent être séparées par une virgule.

Les chaînes données comme valeur dans une instruction DATA peuvent être ou non placées entre guillemets. Cependant, les guillemets sont obligatoires lorsqu'une valeur de chaîne contient une virgule, deux points (:) ou un espace significatif de début ou de fin.

COMMENTAIRES

Des valeurs numériques et des valeurs de chaînes peuvent figurer dans une même instruction DATA. Les valeurs contenues dans l'ensemble des instructions DATA d'un programme forment un fichier séquentiel unique (c'est-à-dire une liste ordonnée), quels que soient

le nombre de valeurs par instruction, la nature de ces valeurs et l'emplacement (par rapport aux instructions READ) des instructions DATA dans le programme.

Ce fichier est stocké dans le programme BASIC (et non sur un support mémoire externe) et l'ordinateur déplace un pointeur dans le fichier à mesure que les valeurs sont lues par les instructions READ. Le type de variable (numérique ou chaîne de caractères) spécifié comme argument de l'instruction READ doit correspondre à celui de la valeur courante positionnée par le pointeur.

Les instructions DATA peuvent comporter un plus grand nombre de valeurs que n'ont à en lire les instructions READ lors de l'exécution d'un programme (auquel cas les valeurs supplémentaires sont ignorées). Par contre, l'inverse n'est pas vrai. Si le nombre de variables des instructions READ est supérieur au nombre d'éléments de DATA, un message d'erreur est affiché (''Data exhausted'', plus de données disponibles). Cependant l'instruction RESTORE permet de réinitialiser la position du pointeur au niveau de la première valeur de la première instruction DATA du programme, ou d'une autre ligne d'instruction DATA spécifiée comme paramètre de l'instruction RESTORE.

EXEMPLE DE PROGRAMME

Dans cet exemple, les données sont lues à l'intérieur de boucles FOR/NEXT. Cette méthode peut être utilisée avec profit pour assigner des valeurs à un tableau.

On remarquera en particulier que le tableau de chaîne A$ et le vecteur A sont remplis successivement et non alternativement. Une séquence du type :

```
30 FOR I=1 TO 8
35 READ A$(I),A(I)
40 NEXT I
```

conduirait à une erreur de syntaxe (''Syntax error''). En effet, le fichier de données étant séquentiel, la valeur assignée à A(1) serait SEINE ET MARNE, c'est-à-dire une valeur de chaîne.

```
10 CLS
20 PRINT "VOULEZ-VOUS LA LISTE"
30 PRINT "- DES NOMS (1)"
```

```
40 PRINT "- DES NUMEROS (2)"
50 PRINT "- DES NOMS ET DES NUMEROS (3)"
60 INPUT "DES DEPARTEMENTS DE L'ILE-DE-F
RANCE";W
70 FOR I=1 TO 8: READ A$(I): NEXT I
80 FOR I=1 TO 8: READ A(I): NEXT I
90 LOCATE 1,10
100 ON W GOTO 110,120,130
110 FOR I=1 TO 8: PRINT A$(I): NEXT I:
    END
120 FOR I=1 TO 8: PRINT A(I): NEXT I:
    END
130 FOR I=1 TO 8: PRINT A(I),A$(I): NEXT
    I: END
140 DATA PARIS,SEINE ET MARNE,YVELYNES,E
SSONNE,HAUTS-DE-SEINE,SEINE-ST-DENIS,VAL
-DE-MARNE,VAL D'OISE
150 DATA 75,77,78,91,92,93,94,95
```

L'instruction REM permet d'insérer des commentaires dans le listing d'un programme. Ces commentaires ne sont pas affichés à l'écran et sont ignorés lors de l'exécution du programme.

FORMAT

REM remarque

- **remarque** est constitué d'une suite quelconque de caractères.

COMMENTAIRES

Les instructions REM permettent de documenter un listing de programme avec des informations courtes et explicites. Cependant ces instructions occupent une partie de la mémoire centrale de l'ordinateur. Dans de longs programmes, on est amené à chercher le meilleur compromis possible entre la nécessité de pouvoir disposer d'une place mémoire suffisante pour y stocker programmes et données et le souci d'établir des listings facilement déchiffrables.

L'exécution d'un programme peut être redirigée, au moyen d'un THEN ou d'un GOTO par exemple, vers une ligne commençant par une instruction REM. Dans ce cas, le déroulement du programme reprend à la première instruction exécutable qui suit cette instruction REM.

Les instructions REM ne sont pas nécessairement placées en tête d'une ligne de programme. Lorsqu'elles constituent la dernière (ou la seule) instruction d'une ligne, il est possible de remplacer le mot clé REM par une simple apostrophe. Par exemple, les deux lignes suivantes sont équivalentes :

```
100 PRINT "première instruction":REM dernière instruction

100 PRINT "première instruction" 'dernière instruction
```

La fonction REMAIN désactive le chronomètre spécifié et renvoie le temps qui restait à courir avant l'exécution de l'instruction GOSUB associée à la commande AFTER... ou EVERY...

FORMAT

REMAIN (n° de chronomètre)...

- Le **numéro de chronomètre** peut prendre les valeurs 0, 1, 2 ou 3.

COMMENTAIRES

Si le chronomètre n'a pas été déclenché par une instruction AFTER... ou EVERY..., la fonction REMAIN retourne la valeur 0. Le programme suivant :

```
10 EVERY 50 GOSUB 100
20 FOR I=1 TO 3000 : NEXT I
30 A=REMAIN(0)
40 B=REMAIN(1)
50 PRINT A;";";B
60 END
100 SOUND 2,25
110 RETURN
```

génère l'affichage :

```
38 ; 0
```

On remarque que la fonction REMAIN ne peut pas apparaître seule sur une ligne de commande ou de programme ; le résultat retourné doit être stocké dans une variable.



Commande

La commande RENUM permet de renuméroter les lignes d'un programme.

FORMAT

RENUM < nouveau numéro > < , < ancien numéro >
< ,incrément > >

- **nouveau numéro** est le premier numéro devant être affecté à la première ligne renumérotée. La valeur par défaut est 10.
- **ancien numéro** est le numéro de la ligne de programme où doit commencer la renumérotation. Lorsque cet argument n'est pas précisé, la renumérotation commence à partir de la première ligne du programme.
- **incrément** est l'écart entre les numéros de deux lignes consécutives dans la nouvelle séquence. La valeur par défaut est 10.

COMMENTAIRES

Lorsque la commande RENUM est exécutée, les numéros de lignes figurant dans les instructions GOTO, GOSUB, THEN...ELSE, ON...GOTO, ON...GOSUB, RESTORE, RESUME et ERL sont également modifiés, afin de s'accorder avec la nouvelle numérotation du programme.

La commande RENUM ne peut pas être utilisée pour intervertir l'ordre de certaines lignes d'un programme. Par exemple, la commande RENUM 20,50 exécutée sur un programme comportant les lignes 10,30 et 50 conduira à l'affichage du message d'erreur ''Improper argument'' (Argument incorrect). Le même message apparaît lorsque la renumérotation a pour effet de créer des numéros de ligne supérieurs à 65535.

Exemples

```
RENUM
```

renumérote toutes les lignes de programme, la première ayant désormais le numéro 10. L'incrément entre deux lignes consécutives est de 10.

```
RENUM 100,,20
```

renumérote tout le programme. La première ligne a désormais le numéro 100 et l'incrément entre deux lignes consécutives est de 20.

```
RENUM 100,90,20
```

attribue le numéro 100 à l'ancienne ligne 90 et renumérote les lignes suivantes de 20 en 20.



RESTORE

Instruction

L'instruction RESTORE réinitialise le pointeur du fichier séquentiel formé par les valeurs contenues dans les instructions DATA, au niveau d'une ligne d'instruction spécifiée ou à celui de la première instruction DATA du programme.

FORMAT

RESTORE < N° de ligne >

- Le **N° de ligne** doit correspondre à une ligne existant dans le programme. Le pointeur est alors placé au niveau de la première donnée de la première instruction DATA rencontrée après ce numéro de ligne.

COMMENTAIRES

La lecture du fichier séquentiel constitué par l'ensemble des données contenues dans les instructions DATA du programme est effectuée par les instructions READ à partir de la position courante du pointeur. RESTORE permet de réinitialiser cette position courante. Lorsque le numéro de ligne est omis, le pointeur est repositionné au niveau de la première valeur du fichier.

Le numéro de ligne spécifié ne doit pas nécessairement correspondre à une ligne contenant une instruction DATA. Le pointeur se placera en effet automatiquement au niveau de la première valeur de l'instruction DATA se trouvant au numéro de ligne supérieur le plus proche.

EXEMPLE DE PROGRAMME

Le programme suivant affiche à l'écran la liste des nombres premiers inférieurs à 10, ou compris entre 10 et 20, ou encore compris entre 20 et 30.

La variable B détermine le nombre de données contenues dans chacune des listes. Le pointeur est réinitialisé de manière différente (lignes 200 à 400) selon l'option choisie.

```
10 CLS
20 PRINT "SOUHAITEZ-VOUS LA LISTE DES NO
MBRES      PREMIERS"
30 PRINT " - INFERIEURS A 10 (1)"
40 PRINT " - COMPRIS ENTRE 10 ET 20 (2)"
50 PRINT " - COMPRIS ENTRE 20 ET 30 (3)"
60 INPUT W
70 CLS
80 ON W GOSUB 200,300,400
90 READ A$
100 PRINT A$
110 FOR I=1 TO B
120 READ N
130 PRINT N
140 NEXT I
150 PRINT: GOTO 20
159 '**********************************
160 DATA NOMBRES PREMIERS INFERIEURS A 1
0
165 DATA 1,2,3,5,7
169
170 DATA NOMBRES PREMIERS COMPRIS ENTRE
10 ET 20
175 DATA 11,13,17,19
179 '
180 DATA NOMBRES PREMIERS COMPRIS ENTRE
20 ET 30
185 DATA 23,29
189 '
190 '**********************************
191 'SOUS-PROGRAMMES REINITIALISANT LE
     POINTEUR
192 '**********************************
200 RESTORE 160:B=5:RETURN
300 RESTORE 170:B=4:RETURN
400 RESTORE 180:B=2:RETURN
```

RND et RANDOMIZE

Fonction et instruction

La fonction RND renvoie un nombre aléatoire compris entre 0 et 1. L'instruction RANDOMIZE initialise la fonction RND pour une séquence particulière.

FORMATS

RND < expression numérique >

RANDOMIZE expression numérique

- L'**expression numérique** possède une valeur quelconque, positive ou négative.

COMMENTAIRES

Les nombres générés par la fonction RND sont des nombres pseudo-aléatoires. Le paramètre associé à la commande RANDOMIZE sert à initialiser cette séquence et, si l'initialisation est identique, la suite de nombres aléatoires est elle-même identique.

Si l'on souhaite que l'ordinateur produise toujours la même série, on insère au début du programme une ligne contenant l'instruction RANDOMIZE. Cela est très intéressant pour la mise au point de certains programmes, programmes de jeux par exemple.



La commande RUN déclenche l'exécution du programme se trouvant en mémoire centrale de l'ordinateur. Elle peut aussi, avec une syntaxe particulière, charger un programme se trouvant sur disquette avant de l'exécuter.

FORMATS

RUN < numéro de ligne >

RUN ''nom de programme''

COMMENTAIRES

La seconde syntaxe de la commande RUN est équivalente à l'instruction LOAD utilisée pour charger puis exécuter un programme stocké sur disquette ou sur cassette.

Lorsque, avec la première syntaxe, le numéro de ligne n'est pas précisé, l'exécution commence à la première ligne du programme en mémoire. S'il est spécifié, l'exécution commence à partir de cette ligne.

L'exécution de la commande RUN a pour conséquence d'initialiser toutes les variables à 0.

La commande RUN ''nom de programme'' est la seule qui permette l'exécution d'un programme protégé.



Instruction

L'instruction SAVE sauvegarde un programme sur disquette ou sur cassette.

FORMAT

SAVE ''nom du fichier'' < ,type de fichier >
< ,paramètres binaires >

- Le **type de fichier** peut être A (ASCII), P (protégé) ou B (binaire) ; si aucun paramètre n'est indiqué, il est stocké sous forme codée condensée.

  Si le fichier est sauvegardé sous forme binaire, certains paramètres doivent être spécifiés lors de la sauvegarde : l'adresse du début de zone mémoire à sauvegarder, la longueur du fichier et l'adresse du point d'entrée.
- Le **nom du fichier** doit respecter les règles énoncées dans la rubrique ''Commandes AMSDOS''.

COMMENTAIRES

L'instruction SAVE sauvegarde un programme sur cassette ou sur disquette. Dans le cas d'une sauvegarde sur disquette, un programme préalablement stocké sur celle-ci sous le même nom de fichier est automatiquement réécrit avec l'extension de nom .BAK (voir la rubrique ''Commandes AMSDOS''). La forme sous laquelle est stocké le fichier dépend des paramètres de l'instruction SAVE.

Un programme stocké sous forme binaire ne peut pas être fusionné (au moyen de la commande MERGE) avec un autre programme se trouvant déjà en mémoire centrale de l'ordinateur.

Un fichier sauvegardé par une instruction SAVE sera à nouveau chargé en mémoire centrale au moyen de l'instruction LOAD (ou, dans certains cas, MERGE ou RUN), sauf s'il a été sauvegardé avec l'option P (protégé) ; dans ce cas, seule la commande RUN permet de le charger puis de l'exécuter.

Exemple

```
SAVE "MACHO",A
```

Le programme résidant en mémoire centrale est stocké, sous le nom de MACHO et sous forme ASCII, sur disquette ou sur cassette.

```
SAVE "MAFROI",B,6000,5000,6005
```

Le contenu de la zone mémoire comprise entre les adresses 6000 et 11000 (6000 + 5000) est recopié sur disque ou sur cassette sous le nom "MAFROI" ; le point d'entrée est à l'adresse 6005. Il est possible de sauvegarder de cette manière les 16K de la mémoire écran, sous forme de fichier binaire :

```
SAVE "ECRAN",B,&C000,&3FCF
```

(ENT, ENV, ON SQ...GOSUB, RELEASE, SQ, SOUND)

La programmation des sons est relativement complexe sur Amstrad. Cette complexité est due au fait qu'il est possible de contrôler individuellement presque tous les paramètres formant une onde sonore. Les instructions et fonctions gouvernant la gestion du son vont être détaillées après un bref rappel de notions élémentaires de solfège.

LE SOLFÈGE ET LA REPRÉSENTATION DES NOTES

La gamme musicale est divisée en octaves. La fréquence d'une note appartenant à une octave donnée est égale au double de la fréquence de la note portant le même nom dans l'octave immédiatement inférieure. Par exemple, la note *ré* de l'octave 0 correspond à une fréquence de 293,665 Hz alors que la fréquence de la note *ré* de l'octave 1 vaut 587,330 Hz. La note émise à une fréquence de 440 Hz correspond à la fréquence de vibration du diapason. Cette note de référence est appelée *la* international ; elle définit la note *la* de l'octave 0.

Une convention occidentale divise chaque octave en 12 unités appelées demi-tons. La différence de fréquence entre deux notes consécutives (*do* et *ré*, ou *sol* et *la* par exemple) correspond à deux demi-tons. Les demi-tons définissent l'altération (dièse ou bémol) des notes. Une note dièse est jouée un demi-ton au dessus de la note non altérée, alors que le bémol correspond à une altération d'un demi-ton au-dessous. La fréquence de la note (avec ou sans altération) correspondant à un demi-ton particulier se calcule au moyen de la formule suivante :

$$\text{FREQUENCE} = 440 \times (2 \text{ puissance } N + (10-n) \times 12)$$

- **N** est le numéro de l'octave, l'octave 0 étant celle à laquelle appartient le *la* international. Les numéros sont positifs et négatifs de part et d'autre de cette octave particulière.
- **n** est le numéro du demi-ton dans l'octave.

La correspondance entre le nom des notes et les numéros des demi-tons est la suivante (# représente le dièse, b le bémol) :

DO	DO#	RÉ	MIb	MI	FA	FA#	SOL	SOL#	LA	SIb	SI
1	2	3	4	5	6	7	8	9	10	11	12

L'ensemble des notes (altérées et non altérées) d'une octave forme une *gamme chromatique*. Les altérations ne sont définies que s'il existe une touche noire correspondante sur le clavier du piano. C'est la raison pour laquelle les notes :

MI# (ou FAb) et SI# (ou DOb)

n'existent pas dans la gamme chromatique. La dénomination des notes altérées est purement conventionnelle ; on parle de *do* dièse et non de *ré* bémol pour désigner le demi-ton situé entre le *do* et le *ré*.

La notation anglo-saxonne représente les notes par une lettre unique, et non par une syllabe. La lettre C correspond au *do*, D à *ré*, ..., G à *sol*, A à *la* et B à *si*. Cette notation est utilisée dans le programme de démonstration, afin de simplifier la saisie des notes.

D'un point de vue physique, l'émission d'une note correspond à la propagation d'une onde sonore. Comme les autres types d'ondes, une onde sonore peut être caractérisée par l'un ou l'autre des deux paramètres suivants : sa fréquence ou sa période. La fréquence définit le nombre de cycles par unité de temps, tandis que la période correspond à la durée d'un cycle. La relation entre ces deux paramètres est la suivante (lorsque la fréquence est exprimée en hertz (Hz)) :

PERIODE = 125 000 / FREQUENCE

Les fréquences et les périodes des notes de la gamme chromatique sur huit octaves sont données dans le manuel de l'utilisateur livré avec l'ordinateur (Annexe VII pour le manuel du CPC 464 et Paragraphe 5 du Chapitre 7 pour le manuel du CPC 664).

La fréquence (ou la période) n'est qu'un des éléments qui permettent de caractériser une note ; d'autres paramètres importants doivent être précisés, en particulier sa durée. La durée absolue d'une note se calcule à partir de deux paramètres : la valeur de la note (noire, blanche, etc.) et le tempo du morceau de musique, c'est-à-dire le

nombre de noires émises par minute (la noire représentant la durée unitaire pour le calcul du tempo). Les valeurs relatives des notes sont les suivantes (de la plus courte vers la plus longue) :

Triple croche	1/8 de noire
Double croche	1/4 de noire
Croche	1/2 noire
Noire	
Blanche	2 noires
Ronde	4 noires

La longueur d'une note peut d'autre part être augmentée de moitié si on la fait suivre d'un point. Ainsi, une blanche pointée correspond à 3 noires.

Signalons enfin qu'une onde sonore n'est généralement pas une sinusoïde parfaite. Il existe des variations d'amplitude (c'est-à-dire de volume) et éventuellement des variations rapides de fréquence à l'intérieur d'un cycle (vibrato). Ces deux paramètres peuvent être reproduits par le synthétiseur de l'Amstrad au moyen des commandes ENV (enveloppe de volume) et ENT (enveloppe de ton).

LE SYNTHÉTISEUR DE SON DE L'AMSTRAD

Le synthétiseur de son de l'Amstrad possède trois canaux sonores et un canal de bruit. Huit octaves sont disponibles (– 3 à 4). Les canaux sont indépendants mais peuvent néanmoins être synchronisés si l'on donne une valeur adéquate au premier paramètre de l'instruction SOUND. Au niveau de chaque canal, les sons à jouer sont placés dans une file d'attente (les fameuses *queues* qui attendent des "rendez-vous", en fonction de la position des bits, dans le manuel Amstrad), chaque file d'attente pouvant contenir jusqu'à cinq notes. La gestion de ces files peut être étudiée simplement au moyen des quelques lignes de programme suivantes :

```
10 SOUND 1,284+8*I,300
20 IF I=9 THEN 50 ELSE PRINT I
30 I=I+1
40 GOTO 10
50 END
```

Le paramètre I permet de modifier la période de la note jouée par l'instruction SOUND de la ligne 10. Chaque note a une durée de 3 secondes (paramètre 300). La valeur de I est affichée à chaque modification. Lorsque l'exécution du programme est terminée (ligne 50), le message Ready (prêt) apparaît à l'écran. Il suffit de faire tourner ce programme pour comprendre l'organisation d'une file d'attente.

Dès le début de l'exécution, les valeurs 0 à 4 sont affichées et la première note commence à être jouée. Les cinq premières notes ont donc été placées dès le début du programme dans la file d'attente. Le programme a continué de s'exécuter, sans attendre la fin de l'exécution de la première instruction SOUND. Dès qu'une note a été jouée, la file se décale vers le haut et une nouvelle valeur de I s'affiche (c'est-à-dire qu'une nouvelle note prend place en fin de file). La fin de l'exécution du programme (message Ready) se termine bien avant que la dernière note ait été jouée, la file devant encore être vidée.

Cette organisation permet d'optimiser le temps d'utilisation du microprocesseur. Les opérations qu'il doit réaliser s'exécutent beaucoup plus rapidement que la durée habituellement programmée pour une note. Aussi est-il possible, au moyen de la fonction SQ et de l'instruction ON SQ(n)...GOSUB, de simuler une utilisation multitâche, c'est-à-dire de faire jouer en continu un morceau de musique tandis que le microprocesseur paraît effectuer une opération complètement différente (affichage, calcul, impression, etc.).



Instruction

L'instruction SOUND permet de programmer les sons produits par le circuit générateur intégré à l'Amstrad.

FORMAT

SOUND sélection du canal,période du ton, < durée > ,
< volume > , < enveloppe du volume > ,
< enveloppe du ton > , < période d'un bruit >

- Seuls les deux premiers paramètres doivent être obligatoirement fournis, les autres étant optionnels.

PARAMÈTRES

Sélection du canal (et synchronisation)

Ce paramètre est codé sur un octet et peut donc recevoir une valeur comprise entre 0 et 255. Il sert à diriger le son vers l'un des trois canaux du synthétiseur, ainsi qu'à établir les conditions de synchronisation avec les sons transmis par les autres canaux. Le positionnement de chacun des huit bits de cet octet correspond à une action particulière :

N° du bit	Poids (décimal)	Commande lorsque le bit est positionné
0	1	Dirige le son vers le canal A
1	2	Dirige le son vers le canal B
2	4	Dirige le son vers le canal C
3	8	Synchronisation avec le canal A
4	16	Synchronisation avec le canal B
5	32	Synchronisation avec le canal C
6	64	Bit d'attente
7	128	Bit de priorité

Le positionnement du bit 6 (bit d'attente) provoque l'interruption de la transmission des sons par le canal sélectionné par l'instruction

SOUND correspondante. Les notes se trouvant dans la file sont en attente jusqu'à ce que soit exécutée une instruction RELEASE(n), n spécifiant le numéro du canal (1 pour A, 2 pour B et 4 pour C).

Le positionnement du bit 7 déclare que le son correspondant à cette instruction SOUND est prioritaire sur tout autre son en attente dans la file ou en train d'être transmis par l'intermédiaire du canal (ou des canaux) utilisé(s) par l'instruction prioritaire. Quand une telle instruction SOUND prioritaire est rencontrée, tous les sons se trouvant dans la file correspondante sont annulés et celui qui est spécifié par l'instruction prioritaire est aussitôt transmis. Une telle instruction peut par exemple être utilisée pour interrompre une musique de fond et la remplacer par un bruit (explosion, etc.) lorsqu'un événement particulier se produit (superposition de deux figures à l'écran, etc.).

La synchronisation entre deux notes jouées par l'intermédiaire de canaux différents (A et B par exemple) nécessite que :

- la note jouée par le canal A ait son indicateur de synchronisation avec le canal B (bit 4) positionné ;
- la note jouée par le canal B ait son indicateur de synchronisation avec le canal A (bit 3) positionné.

Dans ces conditions, la file d'attente associée, selon le cas, à l'un ou l'autre des canaux sera interrompue jusqu'à attendre l'arrivée, en tête de l'autre file, de la note devant être synchronisée.

La valeur à donner au paramètre de sélection du canal se calcule simplement en additionnant les valeurs décimales correspondant aux bits que l'on veut positionner en fonction de l'action souhaitée. Par exemple, l'instruction :

```
SOUND 17,338,300
```

génère une note jouée par le canal A et synchronisée avec une note provenant du canal B. Certaines valeurs données au paramètre de sélection du canal peuvent provoquer des incohérences. Par exemple, la valeur 9 correspond à l'émission d'une note via le canal A, qui doit être synchronisé avec lui-même.

Période de ton

Ce paramètre permet de sélectionner la période de la note émise (ou plus précisément celle du demi-ton, voir ci-dessus). La période

est un nombre entier positif compris entre 0 et 4095. Les périodes des demi-tons (et les fréquences correspondantes) sont données dans le manuel de l'utilisateur livré avec l'ordinateur. La valeur 0 permet de sélectionner le canal de bruit.

Durée

La durée est un paramètre optionnel dont la valeur doit être comprise entre −32768 et 32767. Les valeurs positives correspondent à des centièmes de seconde. La valeur par défaut est 20, soit une durée de deux dixièmes de seconde. La valeur 0 indique que la durée est spécifiée par la commande d'enveloppe de volume (ENV). De même, une valeur négative indique le nombre de répétitions de l'enveloppe de volume. Pour une enveloppe correspondant à une durée de 10 centièmes de seconde (voir l'instruction ENV), la valeur −8 donnée au paramètre "durée" de l'instruction SOUND permettra d'exécuter huit fois la commande ENV, soit une durée totale de 80 centièmes de seconde.

Lorsqu'une valeur positive est donnée au paramètre "durée" de l'instruction SOUND et qu'une enveloppe de volume est en même temps sélectionnée, la durée spécifiée par l'instruction SOUND prime sur celle qui correspond à l'enveloppe de volume, en cas de non-identité des deux valeurs (voir le tableau donné en illustration de l'instruction ENV).

Volume

La valeur de ce paramètre peut être comprise entre 0 et 7 (valeur par défaut : 4) si aucune enveloppe de volume n'est sélectionnée, ou entre 0 et 15 (valeur par défaut : 12) si une enveloppe de volume est précisée. Le niveau du son peut être également réglé manuellement à l'aide de la molette située sur le côté droit du boîtier de l'Amstrad.

Numéro de l'enveloppe de volume

Ce numéro doit être compris entre 0 et 15. La valeur par défaut est 0, signifiant qu'aucune enveloppe de volume n'est sélectionnée (voir l'instruction ENV).

Numéro de l'enveloppe de ton

Quinze enveloppes de ton différentes peuvent être définies au moyen d'instructions ENT. La valeur de ce paramètre (entre 0 et 15) permet d'en sélectionner une. La valeur par défaut est égale à 0, c'est-à-dire qu'aucune enveloppe n'est sélectionnée.

Période du bruit

La valeur de ce paramètre optionnel peut être comprise entre 0 et 31. Un bruit ''pur'' est produit en donnant la valeur 0 au second paramètre de l'instruction SOUND et une valeur comprise entre 1 et 31 à ce dernier paramètre. Il est cependant possible de combiner un bruit avec des notes ou des tons plus conventionnels : il suffit de spécifier une valeur non nulle pour les deux paramètres ''période de ton'' et ''période de bruit''.

L'instruction ENV permet de sélectionner une enveloppe de volume.

FORMAT

ENV N° de l'enveloppe < ,section 1 > < ,section 2 > ...
< ,section 5 >

- Le **N° d'enveloppe** doit être compris entre 1 et 15. Ce paramètre permet de sélectionner une enveloppe particulière.
- Les **sections** déterminent la forme de l'enveloppe. Une enveloppe peut contenir jusqu'à cinq sections. Chaque section comporte trois paramètres séparés les uns des autres par une virgule. Les trois paramètres sont les suivants :

  Le nombre de pas de la section (entre 0 et 239).

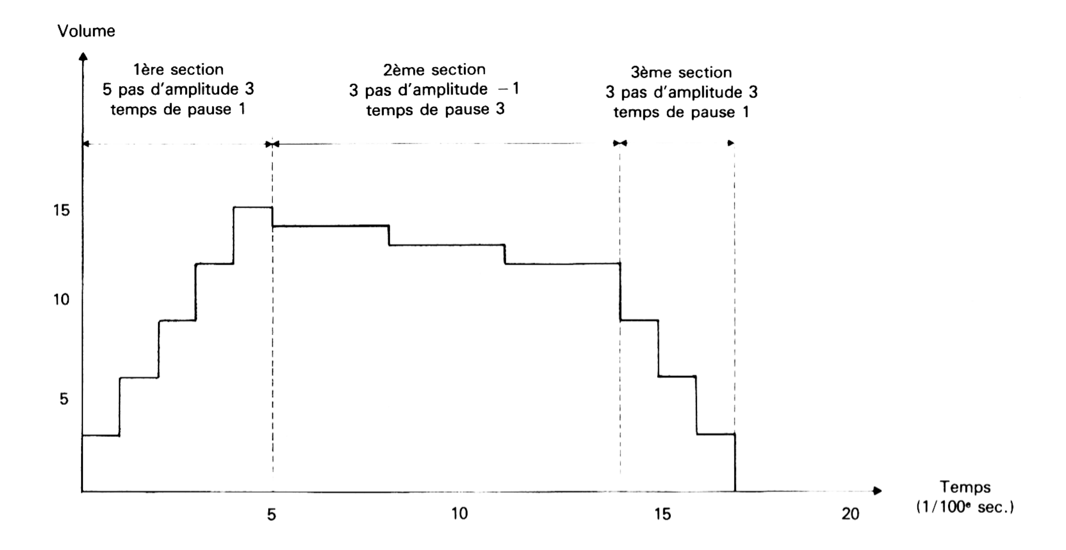
  L'amplitude (volume) associée à chaque pas (entre −15 et +15). Une amplitude nulle correspond à la valeur 0. Une valeur positive spécifie une augmentation d'amplitude, une valeur négative une diminution de cette amplitude.

  Le temps de pause, c'est-à-dire la durée du pas. Le temps de pause doit être compris entre 0 et 255, une unité correspondant à un centième de seconde.

Une note jouée au piano est généralement assez bien reproduite si l'on utilise trois sections (attaque, tenue et chute de la note). L'attaque correspond à l'augmentation de volume résultant de la frappe de la touche. Le volume baisse ensuite lentement (tenue) pour s'éteindre rapidement lorsque l'on relâche la touche (chute). L'instruction ENV correspondante (enveloppe 1) pourra être la suivante :

```
ENV 1,5,3,1,3,-1,3,3,3,1
```

La forme d'une telle enveloppe est représentée sur la figure ci-dessous.



La durée totale spécifiée par l'enveloppe ainsi définie correspond à 17 centièmes de seconde ($(5 \times 1) + (3 \times 3) + (3 \times 1)$). Lorsque cette durée totale (qui représente la somme pondérée des temps de pause qu'elle contient) diffère de la durée spécifiée par le troisième paramètre de l'instruction SOUND associée, c'est la durée spécifiée par SOUND qui est généralement prépondérante. Il existe néanmoins quelques exceptions. Le tableau suivant examine les différentes situations qui peuvent se présenter (S représente la valeur du paramètre "durée" de l'instruction SOUND et E représente la somme pondérée des temps de pause de l'enveloppe de volume sélectionnée) :

$S = E$	Durée S (donc également E).
$S < E$ ($S > 0$)	Durée S. L'instruction ENV n'est pas complètement exécutée.
$S > E$	Durée S. Le volume demeure constant durant les $(S - E)$ derniers centièmes de seconde.
$S = 0$, $E > 0$	Durée E.
$S < 0$, $E > 0$	Durée SV. L'instruction ENV est exécutée S fois.

Lorsqu'une instruction ENV est exécutée, il convient de veiller à ce que la valeur du paramètre de volume reste positive sans qu'elle

excède 15. Il faut en effet savoir que cette valeur est cumulative (le volume spécifié pour le deuxième pas s'ajoute algébriquement à celui du premier, etc.). Lorsque le paramètre d'amplitude atteint la valeur 16, il est automatiquement remis à 0. Cette remise à 0 s'effectue pour chaque valeur multiple de 16 (entre −128 et +127).



L'instruction ENT (enveloppe de ton) permet d'introduire une variation de fréquence (vibrato) dans la note jouée.

FORMAT

ENT N° de l'enveloppe < ,section 1 > < ,section 2 > ...
< ,section 5 >

- Les paramètres ont la même signification que celle qui est donnée à propos de l'instruction ENV. Là encore, chaque section peut posséder trois paramètres, l'amplitude de l'instruction ENV devant simplement être ici remplacée par l'incrément de période de ton (valeur comprise entre −128 et +127), les valeurs négatives correspondant à une diminution de la période.
  Il est cependant possible de ne spécifier que deux paramètres pour chaque section. Ceux-ci ont alors la signification suivante :

  Premier paramètre : *période de ton*. Une nouvelle valeur absolue est alors sélectionnée pour la période de ton (voir le second paramètre de l'instruction SOUND).

  Second paramètre : *temps de pause*, l'unité étant le centième de seconde et la valeur associée pouvant être comprise entre 0 et 255.

Un vibrato s'utilise habituellement pour simuler un instrument à cordes ou à vent, ou encore le son produit par une corde vocale. La percussion d'un marteau sur une corde de piano ne produit pas de vibrato.

La fonction SQ retourne l'état du canal sonore spécifié.

FORMAT

SQ(N° de canal)

- Le **numéro de canal** peut prendre la valeur 1, 2 ou 4.

La valeur retournée par la fonction SQ est comprise entre 0 et 255. Elle correspond à la valeur décimale d'un octet indicateur dont le positionnement des bits a la signification suivante :

Bits 0, 1 et 2	Nombre d'entrées disponibles dans la file d'attente du canal spécifié.
Bit 3	Canal synchronisé avec le canal A.
Bit 4	Canal synchronisé avec le canal B.
Bit 5	Canal synchronisé avec le canal C.
Bit 6	Canal en attente en début de file.
Bit 7	Canal en activité.

L'instruction ON SQ...GOSUB provoque un branchement vers un sous-programme lorsqu'une entrée est libre dans la file d'attente du canal spécifié.

FORMAT

ON SQ(N° de canal) GOSUB N° de ligne

- Le **N° de canal** doit être compris entre 1 et 4 (1 pour le canal A, 2 pour le canal B, 4 pour le canal C).



L'instruction RELEASE libère une attente spécifiée par une instruction SOUND.

FORMAT

RELEASE nombre entier

- Le **nombre entier** doit être compris entre 1 et 7 :

1	libération d'une attente sur le canal A
2	libération d'une attente sur le canal B
3	libération d'une attente sur les canaux A et B
4	libération d'une attente sur le canal C
5	libération d'une attente sur les canaux A et C
6	libération d'une attente sur les canaux B et C
7	libération d'une attente sur les canaux A, B et C

EXEMPLE DE PROGRAMME

Ce programme d'illustration permet de jouer n'importe quel morceau de musique, les notes devant simplement être entrées en DATA dans le format suivant :

XO,D,S

X Nom anglo-saxon de la note (une lettre majuscule de l'alphabet, entre A et G).

O Numéro de l'octave dans laquelle doit être jouée la note (chiffre compris entre 1 et 8).

D Chiffre représentant la durée de la note. La durée effective de la note jouée sera obtenue (par le programme) en additionnant la valeur de D et celle du paramètre TEMPO (fixée

à 10 dans notre exemple, mais pouvant être modifiée à loisir par l'utilisateur).

S Paramètre de synchronisation, spécifiant avec quel canal la note doit (éventuellement) être synchronisée.

Chaque ligne de DATA doit se terminer par une chaîne "", en lieu et place d'un paramètre de synchronisation. Lorsque l'on veut spécifier un silence, la lettre indiquant le nom de la note doit être remplacée par R. Les données correspondant au canal A doivent être écrites dans des instructions DATA dont les numéros de lignes sont compris entre 5000 et 5399. Celles qui correspondent au canal B doivent être écrites entre les lignes 5400 et 5699 et celles qui correspondent au canal C ont des numéros commençant à la ligne 5700. Les branchements sont effectués par le programme aux lignes 5000, 5400 et 5700. La première instruction DATA de chaque canal doit donc être écrite à ce niveau. Pour chacun des trois canaux, la dernière instruction DATA doit s'écrire (symbole de terminaison pour le programme) :

```
DATA "",0," "
```

Dans l'exemple donné comme application de ce programme, seules les données correspondant au canal A ont été écrites. Le paramètre de synchronisation (lettre C) est donc inopérant dans ce cas. Pour pouvoir jouer sur les trois canaux, l'apostrophe située en tête de la ligne 30 doit être supprimée. Ces données reproduisent le thème de la chanson enfantine *Ah ! vous dirai-je, Maman*, sur lequel W.A. Mozart a composé plusieurs variations (K 265). Ce morceau est écrit en *ut* majeur et n'utilise donc pas d'altération (dièse et bémol).

```
5 CLS
10 FAIT(1)=-1:FAIT(2)=-1:FAIT(4)=-1
20 NBVOI=3: TEMPO=10
30 'SYNCHA=8: SYNCHB=16: SYNCHC=32
35 LOCATE 1,1
40 PRINT "Attendez ! Je cherche le LA ..
."
50 GOSUB 1000: '--> INITIALISATION ***
60 ON 4-NBVOI GOTO 70,80,90
70 ON SQ(4) GOSUB 4100
80 ON SQ(2) GOSUB 4050
90 ON SQ(1) GOSUB 4000
```

→

```
95 IF FAIT(1) AND FAIT(2) AND FAIT(4) TH
EN 100 ELSE 95
100 INPUT "ONE MORE TIME, LEON (OUI=O)";
Z$
120 IF UPPER$(Z$)<>"O" THEN END ELSE FAI
T(1)=0:FAIT(2)=0:FAIT(4)=0:NOTE(1)=0:NOT
E(2)=0:NOTE(4)=0:GOTO 60
999 '****************************
1000 '     INITIALISATION
1001 '****************************
1010 DIM PERIOD (4,200),DUR (4,200),SYNC
H(4,200)
1020 FOR J=1 TO NBVOI
1030 CANAL=2^(J-1): NUMNOTES=0: FAIT(CAN
AL)=0
1060 IF CANAL=1 THEN RESTORE 5000 ELSE I
F CANAL=2 THEN RESTORE 5400 ELSE IF CANA
L=4 THEN RESTORE 5700
1100 READ NOTE$,DUREE,SYNCH$
1110 IF NOTE$="" THEN 1500
1120 GOSUB 2000: ' --> CALCUL PERIODE **
1200 NUMNOTES=NUMNOTES+1
1210 PERIOD(CANAL,NUMNOTES)=PERIOD
1220 DUR(CANAL,NUMNOTES)=DUREE+TEMPO
1230 SYNCH(CANAL,NUMNOTES)=SYNCH
1240 GOTO 1100
1500 PERIOD(CANAL,0)=NUMNOTES
1510 NEXT J
1600 ENT -1,1,1,7,1,-1,7
1610 RETURN
1999 ' ****************************
2000 '     CALCUL OCTAVE ET NOTE
2001 ' ****************************
2010 Z$=LEFT$(NOTE$,1)
2020 NOTE=INSTR("C D EF G A BR",Z$)-1
2025 IF NOTE=12 THEN OCTAVE=0:GOTO 2100
2030 IF NOTE<0 THEN OCTAVE=5: DUR=1: RET
URN
2040 Z$=RIGHT$(NOTE$,1)
2050 IF Z$="#" THEN NOTE=NOTE+1 ELSE IF
Z$="b" OR Z$="B" THEN NOTE=NOTE-1
2080 OCTAVE=VAL(MID$(NOTE$,2,1))-3
2090 IF NOTE<0 THEN NOTE=0: OCTAVE=OCTAV
E-1
2100 GOSUB 3000
2110 SYNCH=0
2120 FOR P=1 TO LEN(SYNCH$)
2130 IF MID$(SYNCH$,P,1)="A" THEN SYNCH=
SYNCH+SYNCHA
2140 IF MID$(SYNCH$,P,1)="B" THEN SYNCH=
```

```
SYNCH+SYNCHB
2150 IF MID$(SYNCH$,P,1)="C" THEN SYNCH=
SYNCH+SYNCHC
2160 NEXT P
2170 RETURN
2999 ' ********************************
3000 '          CALCUL DE LA PERIODE
3001 ' ********************************
3005 IF NOTE=12 THEN PERIOD=0:RETURN
3010 FREQ=261.626*(2^(OCTAVE+NOTE/12))
3020 PERIOD=ROUND(125000/FREQ):RETURN
4000 CANAL=1:GOTO 4200
4050 CANAL=2:GOTO 4200
4100 CANAL=4
4200 NOTE(CANAL)=NOTE(CANAL)+1
4205 VOL=9:TONTON=2
4210 SOUND CANAL+SYNCH(CANAL,NOTE(CANAL)
),PERIOD(CANAL,NOTE(CANAL)),DUR(CANAL,NO
TE(CANAL)),VOL,0,TONTON
4220 IF NOTE(CANAL)>=PERIOD(CANAL,0) THE
N FAIT(CANAL)=-1:RETURN
4300 J=CANAL:IF J>3 THEN J=3
4310 ON J GOTO 4400,4420,4440
4400 ON SQ(1) GOSUB 4000
4410 RETURN
4420 ON SQ(2) GOSUB 4050
4430 RETURN
4440 ON SQ(4) GOSUB 4100
4450 RETURN
5000 DATA C5,8,C,R,2,,C5,8,C,R,2,,G5,8,C
,R,2,,G5,8,C,R,2,,A5,8,C,R,2,,A5,8,C,G5,
8,C,R,2,,G5,8,C,R,2,,F5,8,C,R,2,,F5,8,C,
R,2,,E5,8,C,R,2,,E5,8,C,R,2," "
5010 DATA D5,8,C,R,2,,D5,6,C,R,1,,E5,2,C
,R,1,,C5,16,C,R,1," "
5020 DATA G5,8,C,R,2,,G5,8,C,R,2,,F5,8,C
,R,2,,F5,8,C,R,2,,E5,8,C,R,2,,E5,8,C,R,2
,,D5,8,C,R,2,,D5,8,C,R,2,,G5,8,C,R,2,,G5
,8,C,R,2," "
5030 DATA F5,8,C,R,2,,F5,8,C,R,2,,E5,2,C
,R,2,,F5,2,C,E5,2,C,D5,2,C,E5,6,C,R,1,,F
5,2,C,R,1,,E5,8,C,R,2,,D5,8,C,R,2," "
5040 DATA C5,8,C,R,2,,C5,8,C,R,2,,G5,8,C
,R,2,,G5,8,C,R,2,,A5,8,C,R,2,,A5,8,C,G5,
8,C,R,2,,G5,8,C,R,2,,F5,8,C,R,2,,F5,8,C,
R,2,,E5,8,C,R,2,,E5,8,C,R,2," "
5050 DATA D5,2,C,R,2,,E5,2,C,D5,2,C,C5,2
,C,D5,6,C,R,1,,E5,2,C,R,1,,C5,16,C,R,1,"
 "
5399 DATA "",0," "
```
→

```
5400 ' notes pour le canal C *********
5500 DATA "",0," "
5700 ' notes pour le canal C *********
5900 DATA "",0," "
```

La fonction SPACE$ fournit une chaîne formée de n espaces.

FORMAT

SPACE$(n)

COMMENTAIRES

La valeur retournée par cette fonction est une chaîne de caractères ; elle doit donc être stockée, si besoin est, dans une variable de chaîne.



Fonction

La fonction SPC génère, à l'écran ou sur l'imprimante, le nombre d'espaces correspondant à la valeur de son argument.

FORMAT

SPC(N)

- **N** est une expression numérique entière.

COMMENTAIRES

Cette fonction n'est utilisable qu'avec l'instruction PRINT. Elle est très utile, en liaison avec la fonction TAB, pour afficher ou imprimer des données présentées sous forme de tableau.

EXEMPLE

```
10 PRINT SPC(5) "Guide du BASIC"
20 PRINT SPC(9) "AMSTRAD"
```

L'expression "Guide du BASIC" sera affichée à partir de la sixième colonne. "AMSTRAD" s'écrira sur la ligne suivante, en commençant à la dixième colonne.

L'instruction SPEED INK initialise les durées d'apparition des couleurs intermittentes associées aux commandes BORDER et INK.

FORMAT

SPEED INK n1, n2

- **n1** et **n2** sont des nombres entiers qui représentent respectivement les durées d'affichage de la première et de la seconde couleurs ; ces durées sont exprimées en 1/50 de seconde.

EXEMPLE DE PROGRAMME

Le programme suivant montre l'effet produit par des variations des paramètres de l'instruction SPEED INK :

```
10 CLS
20 BORDER 24,6
30 FOR I=20 TO 100 STEP 20
40 FOR J=20 TO 100 STEP 20
50 SPEED INK I,J
60 FOR K=1 TO 5000 : NEXT K
70 NEXT J
80 NEXT I
```

SPEED KEY

Instruction

L'instruction SPEED KEY modifie les paramètres contrôlant les réponses des touches du clavier.

FORMAT

SPEED KEY n1, n2

- **n1** représente le temps écoulé entre l'instant où la touche est pressée et celui où elle passe en mode répétition.
- **n2** représente la période de la répétition.

COMMENTAIRES

Lorsqu'une touche du clavier est activée, un caractère est affiché sur l'écran ; si la pression est maintenue sur la touche, le caractère est répété. Les paramètres n1 et n2 sont exprimés en 1/50 de seconde (les valeurs par défaut sont 10 et 10) et peuvent prendre des valeurs comprises entre 1 et 255. La commande :

```
SPEED KEY 1,1
```

met en évidence l'effet de l'instruction.

L'instruction SPEED WRITE définit la vitesse d'enregistrement des données pour un stockage sur cassette.

FORMAT

SPEED WRITE n

- **n** peut prendre la valeur 0 ou 1.

COMMENTAIRES

Deux vitesses de stockage sont disponibles : 1 000 bauds (1 000 bits par seconde), qui correspond à la valeur 0 et 2 000 bauds, qui correspond à la valeur 1.

Quels que soient les paramètres de stockage, l'Amstrad établit automatiquement la vitesse correcte sans intervention de l'utilisateur. La valeur par défaut est 1 000 bauds.

L'instruction STOP arrête l'exécution d'un programme et provoque le retour au mode commande.

FORMAT

STOP

COMMENTAIRES

Les instructions STOP peuvent être insérées après n'importe quelle autre instruction d'un programme. Elles en arrêtent l'exécution et le message suivant est alors affiché :

```
Break in xxxxx
(arrêt en xxxxx)
```

xxxxx étant le numéro de la ligne où l'instruction STOP a été rencontrée. A l'inverse de l'instruction END, STOP ne ferme pas les fichiers, de sorte que l'exécution normale du programme peut reprendre là où elle s'est arrêtée. Pour ce faire, il suffit de taper la commande CONT qui signifie au programme de continuer (voir la description de cette commande). Les trois utilisations principales de l'instruction STOP sont les suivantes :

1. Lors de la mise au point d'un programme, il est souvent utile d'insérer quelques instructions STOP en certains points de celui-ci, afin de vérifier que son exécution est correcte ou de localiser, le cas échéant, l'étape défectueuse. Puisqu'on est alors ramené en mode commande, il est possible d'afficher la valeur de paramètres ou de variables à cette étape du programme, en tapant au clavier la commande PRINT suivie du nom de la ou des variables en question.
2. Lorsqu'un grand nombre d'informations ou de résultats doivent être affichés successivement à l'écran, leur défilement est parfois

trop rapide pour qu'ils puissent être lus par le programmeur. Par exemple, la séquence d'instructions suivante affiche la racine carrée des 100 premiers nombres, en allant chaque fois à la ligne :

```
10 FOR X=1 TO 100
20 PRINT SQR(X)
30 NEXT X
```

Ce calcul étant très rapide et l'écran disposant de moins de 100 lignes d'affichage, les nombres défileraient sans pouvoir être lus. Cela peut être évité en ajoutant la ligne suivante :

```
25 IF X/10 - FIX(X/10) = 0 THEN STOP
```

qui provoquera l'arrêt de l'exécution du programme pour chaque valeur de X multiple de 10.

3. Pour permettre la mise sous tension d'une unité périphérique qui ne l'est pas nécessairement et qui est adressée directement par le programme (par exemple une imprimante).

STRING$

Fonction

La fonction STRING$ renvoie une chaîne formée de N fois le même caractère.

FORMATS

A$ = STRING$(N,X$)

A$ = STRING$(N,X)

- **N** est une expression numérique comprise entre 0 et 255.
- **X$** est une chaîne de caractères dont seul le premier sera utilisé par la fonction.
- **X** est une expression numérique comprise entre 0 et 255 dont la valeur correspond au numéro de code ASCII du caractère qui sera stocké N fois dans la variable A$.

COMMENTAIRES

Cette fonction permet de définir une chaîne de caractères homogène, en spécifiant soit directement ce caractère (premier format), soit son numéro de code ASCII (second format). Une application possible de la fonction STRING$ consiste à définir une chaîne traçant le cadre d'un tableau. Lorsque la chaîne X$ correspond à un espace, les fonctions SPACE$ et STRING$ sont équivalentes, comme dans l'exemple suivant :

```
SPACE$(10)
STRING$(10,"")
```

Les deux instructions suivantes sont équivalentes :

```
PRINT STRING$(10,"*")
PRINT STRING$(10,42)
```

Elles conduisent à l'affichage de :

```
************
```

EXEMPLE DE PROGRAMME

```
5 REM **** PROGRAMME STRING$ ****
10 CLS
15 LOCATE 1,2
20 A$="-"
30 PRINT STRING$(16,A$)"MENU"STRING$(16,
A$)
40 PRINT STRING$(36,A$)
50 PRINT
60 PRINT "Choucroute mayonnaise"
70 PRINT "Petit sale aux lentilles"
80 PRINT "Pommes de terre en robe de cha
mbre "STRING$(1,"*")
90 PRINT "Nouilles garnies "STRING$(2,"*
")
100 PRINT
110 PRINT STRING$(36,A$)
120 LOCATE 2,12
130 PRINT STRING$(1,"*")" A certaines he
ures seulement"
135 LOCATE 2,13
140 PRINT STRING$(2,"*")" Selon la fanta
isie du patron"
150 LOCATE 1,23
```

```
----------------MENU----------------
------------------------------------

Choucroute mayonnaise
Petit sale aux lentilles
Pommes de terre en robe de chambre *
Nouilles garnies **

------------------------------------

 * A certaines heures seulement
 ** Selon la fantaisie du patron
```



La fonction STR$ convertit une expression numérique en une chaîne de caractères.

FORMAT

STR$(n)

- **n** est une expression numérique.

COMMENTAIRES

Si n est un nombre négatif, il est retourné tel quel (avec son signe) par la fonction STR$. S'il s'agit d'un nombre positif dans lequel le signe + n'est pas précisé, un espace est automatiquement réservé dans la chaîne, à gauche du nombre. Par exemple, l'instruction :

```
PRINT STR$(8.25),LEN(STR$(8.25))
```

conduit à l'affichage de :

```
8.25          5
```

Le nombre 8.25 est précédé d'un espace ; la chaîne retournée par la fonction STR$ comprend donc cinq caractères.

La fonction STR$ permet d'effectuer sur des nombres des opérations habituellement réservées aux chaînes. La fonction VAL effectue la conversion inverse (elle transforme une chaîne en un nombre).

L'instruction SYMBOL définit un caractère dont le numéro est spécifié suivant une matrice de 8 points sur 8. L'instruction SYMBOL AFTER indique la limite inférieure des caractères redéfinissables.

FORMATS

SYMBOL n° du caractère, liste d'octets

SYMBOL AFTER limite inférieure

- Le **numéro du caractère** doit être une valeur comprise entre la limite inférieure définie par l'instruction SYMBOL AFTER et 255.
- Les caractères sont inscrits sur une matrice de 8 points sur 8, c'est-à-dire 8 rangées constituées d'un octet chacune ; la liste d'octets comprend donc 8 valeurs séparées par des virgules.
- La **limite inférieure** est une valeur comprise entre 0 et 256 ; la valeur par défaut est 240.

COMMENTAIRES

Le BASIC Amstrad donne à l'utilisateur la possibilité de redéfinir des caractères ; le paramètre associé à SYMBOL AFTER a par défaut la valeur 240, c'est-à-dire qu'il est possible de redéfinir les caractères ayant pour codes les nombres 240 à 255 (16 caractères). Si ce nombre est insuffisant, il suffit de donner à l'instruction SYMBOL AFTER un paramètre qui diminue la limite inférieure : par exemple, pour redéfinir 100 caractères, il est possible d'écrire :

```
10 SYMBOL AFTER 156
```

Ainsi, les caractères dont les codes sont compris entre 156 et 255 seront redéfinissables.

Chaque caractère est représenté par une matrice de 8×8 points ; les points qui sont allumés ont la valeur 1, les points qui sont éteints

ont la valeur 0. Par exemple, la lettre A du jeu de caractères Amstrad (dont le code ASCII est 65) est définie de la manière suivante :

00011000	24
00111100	60
01100110	102
01100110	102
01111110	254
01100110	102
01100110	102
00000000	0

EXEMPLE DE PROGRAMME

Dans le programme ci-dessous, l'instruction SYMBOL AFTER de la ligne 50 autorise la redéfinition des caractères dont les codes sont compris entre 140 et 255. Les lignes 260 à 310 redéfinissent les caractères ayant les codes 140 à 145.

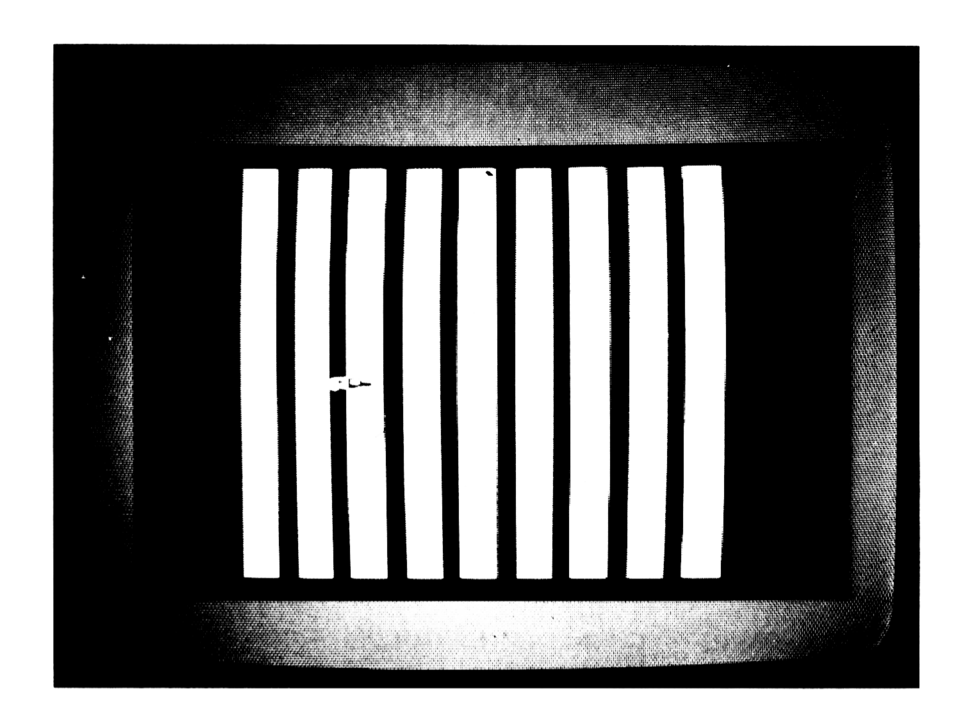
Les caractères 140, 141 et 142 sont concaténés dans la variable VOIT$ pour matérialiser la première et la dernière images de la voiture (lignes 410 et 420) ; il en est de même pour les caractères 143, 144 et 145 qui sont concaténés dans la variable EXORVOIT$.

Les instructions des lignes 400 et 550 sont particulières aux modèles CPC 664 et 6128 ; pour le modèle 464, il faut modifier les lignes de la manière suivante (ces modifications sont détaillées dans l'exemple de programme des instructions MOVE et MOVER) :

```
365 CHR$(23)+CHR$(1)
400 MOVE X2,Y1,2
550 MOVE X2,Y1,2
```

```
10 CLS
20 MODE 1
30 BORDER 7
40 COUL=0
50 SYMBOL AFTER 140
60 GOSUB 150
70 GOSUB 250
80 PEN 1
90 WHILE 1
100 GOSUB 350
110 TAGOFF
```

```
120 PRINT CHR$(7)
130 WEND
140 END
150 '
160 INK 0,1
170 INK 1,24
180 INK 2,20
190 INK 3,24
200 Y=20 : COUL=1
210 FOR X=100 TO 500 STEP 50
220 GOSUB 470
230 NEXT X
240 RETURN
250 '
260 SYMBOL 140,0,15,28,124,127,18,12,0
270 SYMBOL 141,0,255,120,120,255,255,0,0
280 SYMBOL 142,0,128,192,254,254,72,48,0
290 SYMBOL 143,0,16,36,132,128,55,20,0
300 SYMBOL 144,0,0,137,137,0,1,0,0
310 SYMBOL 145,0,128,64,2,2,216,80,0
320 VOIT$=CHR$(140)+CHR$(141)+CHR$(142)
330 EXORVOIT$=CHR$(143)+CHR$(144)+CHR$(1
45)
340 RETURN
350 '
360 TAG
370 X1=0 : Y1=200
380 FOR X2=0 TO 550 STEP 2
390 FRAME
400 MOVE X2,Y1,2,1
410 IF X2=0 THEN PRINT VOIT$; ELSE PRINT
 EXORVOIT$;
420 IF X2=550 THEN GOSUB 540
430 C$=""
440 WHILE C$="" : C$=INKEY$ : WEND
450 NEXT X2
460 RETURN
470 '
480 IF COUL=1 THEN COUL=2 ELSE COUL=1
490 FOR X2=X TO X+30 STEP 2
500 MOVE X2,Y
510 DRAWR 0,360,COUL
520 NEXT X2
530 RETURN
540 '
550 MOVE 550,Y1,2,1
560 PRINT VOIT$;
570 RETURN
```



Un programme destiné à générer des caractères est donné en illustration de l'instruction KEY.

La fonction TAB permet de positionner le curseur ou la tête de l'imprimante par rapport au bord gauche de l'écran ou du papier.

FORMAT

TAB(N)

- **N** est une expression numérique entière. La valeur 1 correspond au bord gauche (colonne 1), la valeur 2 à la deuxième colonne, etc. TAB(0) est identique à TAB(1).

COMMENTAIRES

La fonction TAB ne s'emploie qu'avec l'instruction PRINT. Elle imprime des espaces jusqu'à ce que le numéro de colonne spécifié soit atteint. Cependant, si ce numéro de colonne est inférieur à la position courante du curseur ou à celui de la tête d'imprimante, la fonction TAB exécute d'abord un retour chariot avant de placer le curseur à la position définie. Les chaînes ou les valeurs à imprimer le seront donc à la ligne suivante (par rapport à la position courante du curseur au moment de l'appel de la fonction). Pour une utilisation sur imprimante, la fonction TAB tient compte du format spécifié par l'instruction WIDTH. Pour un affichage à l'écran, la fonction TAB tient compte de la largeur de la fenêtre définie par WINDOW et du mode sélectionné (MODE).

Lorsque la chaîne ou la valeur soumise à la fonction TAB ne peut s'écrire ou s'afficher sur une seule ligne, le curseur se positionne automatiquement au niveau de la première colonne (bord gauche de l'écran).

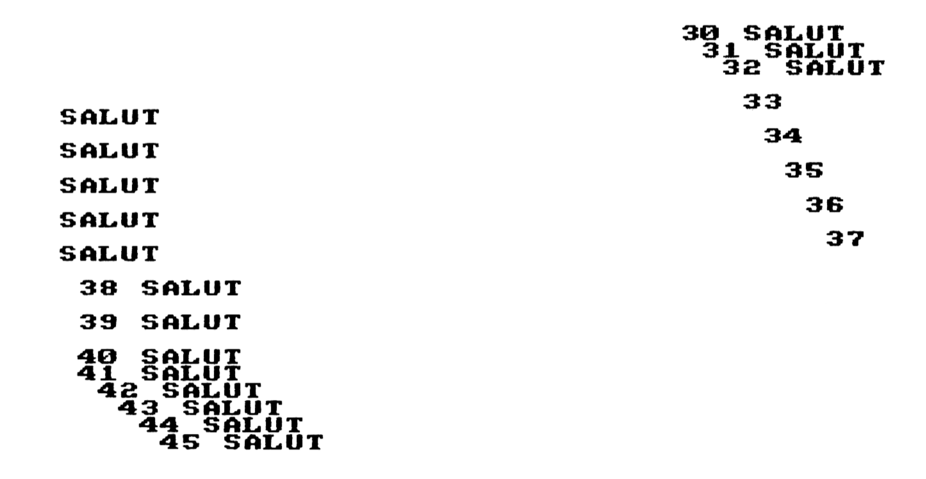
La fonction TAB est particulièrement utile pour créer des tableaux dont les colonnes n'ont pas toutes la même largeur.

Exemple

La manière dont opère la fonction TAB est illustrée par les quelques lignes de programme suivantes :

```
10 MODE 1
20 CLS
30 LOCATE 1,1
40 FOR I=30 TO 45
50 PRINT TAB(I) I;"SALUT"
60 NEXT
```

Le mode 1 sélectionne une largeur d'écran de 40 colonnes. Pour chaque valeur de I comprise entre 30 et 45 doit s'afficher la position de tabulation I gouvernée par TAB(I), suivie du mot SALUT. Pour une valeur de I donnée, I et SALUT doivent être écrits (quand cela est possible) sur une même ligne (puisque I est suivi d'un point-virgule). Par contre, un passage à la ligne est demandé après chaque mot SALUT (puisque cette chaîne n'est pas suivie d'un point-virgule). L'exécution de ce programme conduit à l'affichage suivant :



Pour les valeurs de I comprises entre 30 et 32, I et SALUT peuvent être écrits sur la même ligne et sont donc ainsi affichés. Lorsque I est compris entre 33 et 37, il n'y a plus assez de place pour les écrire tous deux sur la même ligne, à partir de la position de tabulation spécifiée. Il y a par contre suffisamment de place pour écrire la valeur

de I. C'est la raison pour laquelle I s'affiche à la position de tabulation spécifiée, alors que le mot SALUT est écrit à partir de la première colonne de la ligne suivante. A partir de I = 38, il n'y a plus assez de place pour écrire I au niveau de la position spécifiée. La valeur numérique et la chaîne s'inscrivent alors sur la ligne suivante (d'où la ligne vide).

Notez que les nombres 38 à 40 apparaissent à partir de la colonne n° 2 (puisqu'il s'agit de valeurs numériques, une position de caractère est en effet réservée pour le signe, celui-ci n'étant pas affiché dans le cas de valeurs positives). A partir de I = 41, la position de tabulation TAB(I) dépasse le nombre de caractères par ligne du mode en cours (40 colonnes en mode 1). TAB(41) est alors automatiquement converti en TAB(1), TAB(42) en TAB(2), etc.

L'instruction TAG permet d'afficher des caractères (adressés à un canal particulier) sur des positions graphiques de l'écran.
L'instruction TAG OFF annule la commande TAG précédente et affiche les caractères à partir de la position courante du curseur.

FORMATS

TAG < # n° de canal >

TAG OFF < # n° de canal >

- **n° de canal** correspond à une fenêtre texte particulière (0 à 7) ; la valeur par défaut est 0.

COMMENTAIRES

L'instruction TAG permet d'afficher un caractère à chacune des positions du curseur graphique, c'est-à-dire à partir de chaque pixel de l'écran. Il ne faut pas oublier que la taille des pixels varie en fonction du mode (voir l'instruction MODE). Le tableau suivant indique les possibilités de positionnement du curseur texte et du curseur graphique en fonction du mode :

Mode	Positions d'affichage en mode texte	Positions d'affichage en mode graphique
0	25 * 20 = 500	200 * 160 = 32000
1	25 * 40 = 1000	200 * 320 = 64000
2	25 * 80 = 2000	200 * 640 = 128000

EXEMPLE DE PROGRAMME

Le programme illustrant les commandes SYMBOL et SYMBOL AFTER utilise une instruction TAG (ligne 360) pour générer l'affichage

des caractères représentant la voiture à la position en cours du curseur graphique.

Le mode 1 est sélectionné (ligne 20) puis une boucle FOR déplace le curseur graphique de la position 0 à la position 550 avec un pas d'un pixel :

```
380 FOR X2=0 TO 550 STEP 2
```

Il faut se souvenir qu'en mode 1 la taille d'un pixel étant de 2*2 points, pour se déplacer horizontalement d'un pixel, il faut que l'abscisse soit incrémentée de deux points. A chaque itération, une chaîne alphanumérique matérialisant la voiture est affichée à l'écran (ligne 410). Lorsque la largeur de l'écran a été parcourue, le sous-programme se termine et l'instruction TAG OFF replace l'affichage des caractères en mode texte normal.

TEST et TESTR

Les fonctions TEST et TESTR renvoient le numéro de la couleur de l'encre dans laquelle est affiché un point de l'écran dont les coordonnées graphiques absolues (pour TEST) ou relatives (pour TESTR) sont spécifiées.

FORMATS

TEST coordonnée x, coordonnée

TESTR déplacement x, déplacement y

COMMENTAIRES

Les fonctions TEST et TESTR renvoient une valeur qui représente le numéro d'encre du point dont les coordonnées sont spécifiées ; ces valeurs sont donc comprises entre 0 et 15.

La fonction TIME renvoie le temps écoulé depuis la mise sous tension de l'ordinateur ou le temps écoulé depuis la dernière initialisation (RESET).

FORMAT

TIME

COMMENTAIRES

L'ordinateur Amstrad possède une horloge interne qui reçoit une impulsion tous les 1/300 de seconde ; le temps écoulé est donc exprimé en 1/300 de seconde. La commande :

```
PRINT TIME/300
```

affiche le temps en secondes depuis la dernière initialisation du système.

TRON/TROFF

L'instruction TRON active l'affichage des numéros de lignes de programme exécutées. TROFF désactive ce mode.

FORMATS

TRON

TROFF

COMMENTAIRES

L'instruction TRON est particulièrement utile au cours de la mise au point d'un programme. Il est ainsi possible de suivre la logique de son déroulement en examinant le numéro des lignes exécutées (c'est-à-dire en suivant la trace du programme).

Le mode TROFF est le mode en service à la mise sous tension de l'ordinateur. Les instructions TRON et TROFF sont utilisables en mode programme comme en mode direct ; il est à noter que la commande RUN ne désactive pas le mode TRON si celui-ci est en service.

Exemple

```
110 TRON
120 IF A > 0 THEN IF A > 100 THEN 150 ELSE 160 ELSE 170
150 PRINT "A > 100":END
160 PRINT "0 < A < =100":END
170 PRINT "A < =0"
180 TROFF
```

Si A a par exemple la valeur 15, l'affichage suivant sera obtenu :

```
(120) (160) 0 < A < =100
```

signifiant que le programme a exécuté les instructions des lignes 120 et 160.

La fonction VAL retourne les valeurs numériques se trouvant en tête d'une chaîne de caractères donnée comme argument à la fonction.

FORMAT

VAL(n$)

COMMENTAIRES

La valeur retournée est une valeur numérique (elle doit donc être stockée dans une variable de ce type). Les espaces et les tabulations sont automatiquement supprimés par la fonction VAL. Lorsque les premiers caractères de la chaîne n$ ne sont pas des chiffres, la fonction retourne la valeur 0.

Lorsque n$ contient des nombres exprimés en notation binaire ou hexadécimale (donc précédés des préfixes &X ou &H), la valeur retournée est exprimée en décimal.

La fonction STR$ effectue la conversion inverse de celle qui est réalisée par la fonction VAL.

Exemples

```
PRINT VAL("LEO PARRE,720 32 32")
```

La valeur 0 est affichée (il n'y a pas de chiffre en tête de la chaîne).

```
PRINT VAL("720 32 32,LEO PARRE")
```

Le nombre 7203232 est affiché (sans espaces intermédiaires).

```
PRINT VAL("&HE00")
```

Le nombre 3584 (soit E00 en hexadécimal) est affiché.



WAIT

Instruction

L'instruction WAIT suspend l'exécution d'un programme BASIC dans l'attente qu'un port du microprocesseur reçoive une valeur spécifiée.

FORMAT

WAIT n° de port,masque < ,inversion >

- Le **n° de port** est une expression numérique entière comprise entre 0 et 255.
- **masque** est une valeur comprise entre 0 et 255 ; elle est utilisée pour réaliser une opération AND (ET) avec la valeur lue sur le port.
- **inversion** est une valeur comprise entre 0 et 255 ; elle est utilisée pour réaliser une opération XOR (OU exclusif) avec la valeur lue sur le port.

COMMENTAIRES

Cette instruction constitue une boucle interne permettant d'attendre qu'un octet (l'octet 1 de l'instruction) ayant une configuration particulière soit reçu sur le port dont le numéro est spécifié.

La comparaison est effectuée bit à bit au moyen de l'opérateur logique AND (voir la rubrique consacrée aux opérateurs logiques). L'exécution du programme n'est reprise que lorsque le résultat de l'opération :

masque AND octet lu

est différent de 0.

Le paramètre optionnel ''inversion'' permet de tester spécifiquement la position d'un bit de l'octet reçu sur le port. Lorsque cette option est retenue, l'opération logique suivante est réalisée par l'instruction WAIT :

(octet lu XOR inversion) AND masque

L'exécution du programme ne se poursuit que lorsque le résultat de cette opération est différent de 0. L'opérateur XOR a pour effet, dans cette opération, de modifier l'octet lu avant que celui-ci ne soit comparé au ''masque''. Selon la valeur donnée à l'''inversion'', cette option permet de spécifier sur quel(s) bit(s) de l'octet lu doit être effectué le test logique AND.

Les instructions WHILE et WEND définissent une structure de boucle dans le programme. Cette boucle est exécutée tant qu'une condition reste vérifiée.

FORMATS

WHILE expression logique
... instructions
WEND

COMMENTAIRES

Tant que l'expression logique est vraie, le bloc d'instructions compris entre les commandes WHILE et WEND est exécuté.

EXEMPLE DE PROGRAMME

Le programme de gestion de fichier d'adresses illustrant les instructions OPENIN et OPENOUT utilise différentes structures WHILE WEND. A la ligne 240, les instructions WHILE WEND exécutent une boucle tant qu'un caractère n'a pas été tapé au clavier par l'utilisateur :

```
240 WHILE C$="" : C$=INKEY$ : WEND
```

Puis une autre boucle est réitérée tant que le caractère de fin de fichier n'est pas rencontré :

```
300 WHILE NOT EOF
```

Dans cette boucle, une ligne de données est lue dans le fichier "AGENDA" puis assignée à la variable de chaîne T$(INC%). Avant chaque lecture, l'indice du tableau est incrémenté de 1 :

```
310 INC%=INC%+1
320 LINE INPUT #9,T$(INC%)
```



Instruction

L'instruction WIDTH permet de définir le nombre maximal de caractères pouvant être écrits sur chaque ligne par l'imprimante.

FORMAT

WIDTH nombre de caractères

- Le **nombre de caractères** doit être compris entre 1 et 255.

COMMENTAIRES

La valeur par défaut est 132. Lorsque le nombre de caractères spécifiés dans l'instruction a été imprimé, le BASIC envoie les caractères retour chariot (&H0A) et saut de ligne (&H0D) ; si le paramètre de l'instruction WIDTH est 255, ces deux caractères de fin de ligne sont supprimés.

L'instruction WINDOW définit les caractéristiques d'une fenêtre texte sur l'écran pour un canal spécifié.

FORMAT

WINDOW < # n° de canal, > gauche, droite, haut, bas

- **gauche** est le numéro de la première colonne.
- **droite** est le numéro de la dernière colonne.
- **haut** est le numéro de la première ligne.
- **bas** est le numéro de la dernière ligne.

COMMENTAIRES

Le numéro de canal est un nombre compris entre 0 et 7 ; s'il est omis, c'est le canal 0 qui est pris par défaut.

Les numéros de colonnes et de lignes doivent être en accord avec le mode en cours :

Mode	N° de colonne	N° de ligne
0	1 à 20	1 à 25
1	1 à 40	1 à 25
2	1 à 80	1 à 25

EXEMPLE DE PROGRAMME

Ce programme est un utilitaire destiné à faciliter l'écriture d'un autre programme. Il permet en effet de faire défiler (*scrolling*), dans une fenêtre de l'écran, le listing déjà écrit pendant que la suite du programme est en cours d'écriture ou de modification dans une deuxième

fenêtre. Une troisième fenêtre (window #3) est utilisée pour afficher en haut de l'écran (en rouge sur fond jaune) la signification des trois touches fonction associées à cet utilitaire :

- **La touche f0** permet de faire défiler le listing dans la fenêtre supérieure (window #1). Le texte est écrit en noir sur fond vert.
- **La touche Esc** permet d'arrêter le défilement obtenu au moyen de la touche fonction f0. Lorsque l'on appuie une seule fois sur Esc, le défilement est interrompu ; mais il peut être repris si l'on appuie sur n'importe quelle autre touche. Par contre, une seconde activation de la touche Esc interrompt définitivement le défilement en plaçant le curseur dans la fenêtre active de l'écran (fenêtre inférieure). La fenêtre active (window #0) est celle sur laquelle on peut écrire de nouvelles lignes de programme ou bien en modifier d'autres existant déjà. La fenêtre #0 reçoit également tous les messages générés par BASIC. Le texte s'inscrit en jaune sur fond noir, dans cette partie de l'écran.
- **La touche f1** permet de supprimer les fenêtres. La totalité de l'écran redevient ainsi active. Si l'on veut à nouveau travailler en mode multifenêtre, il suffit de taper la commande RUN 1000.

La numérotation des lignes de cet utilitaire commence à la valeur 1000 afin qu'il puisse être facilement fusionné (au moyen de la commande MERGE) avec n'importe quel autre programme.

```
1000 CLS
1010 INK 0,9:INK 1,0:INK 2,24:INK 3,3
1020 MODE 1
1030 WINDOW #1,1,40,5,15
1040 WINDOW #0,1,40,17,25
1050 WINDOW #3,1,40,1,3
1060 PAPER 1:PEN 2
1070 CLS #3
1080 LOCATE 1,3:PAPER #3,2 : PEN #3,3:PR
INT #3,"     f0"SPC(12)"Esc"SPC(11)"f1"SP
C(6);
1090 PRINT #3,"LIST fenetre"  "  ARRET S
crol."  " SUPPRIM fen. "
1100 KEY 128,"cls #1:list ,#1"+CHR$(13)
1110 CLS #0
1120 KEY 129,"window 1,40,1,25:cls:list"
+CHR$(13)
1130 PEN 2:LOCATE 1,13
```

L'instruction WINDOW SWAP permute le contenu des deux fenêtres spécifiées.

FORMAT

WINDOW SWAP n° de canal, n° de canal

- **n° de canal** est une valeur entière comprise entre 0 et 7.

COMMENTAIRES

Cette instruction peut être utilisée au cours d'un programme pour transférer le contenu d'une fenêtre à un autre endroit de l'écran. Il est à noter que le symbole dièse ne doit pas être spécifié dans cette instruction.



WRITE

Instruction

L'instruction WRITE écrit des expressions dans un format particulier, sur le canal spécifié.

FORMAT

WRITE < # n° de canal, > liste d'expressions

- La **liste d'expressions** est une liste de valeurs numériques ou alphanumériques séparées par des virgules ou des points-virgules.

COMMENTAIRES

Le numéro de canal est un nombre compris entre 0 et 9 (8 spécifie l'imprimante tandis que 9 correspond à un fichier ouvert sur cassette ou sur disquette).

La commande WRITE insère des virgules entre les éléments affichés, place les chaînes de caractères entre guillemets et supprime les espaces avant et après les valeurs numériques. Le programme :

```
10 A$="RAT" : B$="D'EAU"
20 A=1267 : B=17.146
30 PRINT A$;B$;A;B
40 PRINT
50 WRITE A$;B$;A;B
```

génère l'affichage suivant :

```
RATD'EAU 1267 17.146
"RAT","D'EAU",1267,17.146
```

Les fonctions XPOS et YPOS renvoient respectivement les coordonnées horizontale et verticale du curseur graphique.

FORMATS

XPOS

YPOS

COMMENTAIRES

Il existe un curseur graphique totalement indépendant du curseur texte. Si le curseur texte peut être ou non affiché selon la valeur des paramètres de l'instruction CURSOR en cours, le curseur graphique n'est quant à lui jamais affiché. Les fonctions POS et VPOS ont la même signification, pour le curseur texte, que celle des fonctions XPOS et YPOS associées au curseur graphique.



L'instruction ZONE permet de modifier les tabulations associées à la commande PRINT.

FORMAT

ZONE expression numérique

- L'**expression numérique** est un nombre compris entre 1 et 255.

COMMENTAIRES

Lorsque la virgule est utilisée comme séparateur dans l'instruction PRINT, celle-ci génère des tabulations de 13 caractères ; si les expressions sont inférieures à 13 caractères, l'affichage débute sur les colonnes 1, 14 et 27 en Mode 1. L'instruction ZONE permet de modifier la valeur de cette tabulation automatique. Par exemple :

```
10 MODE 1
20 ZONE 10
30 A$="*"
40 PRINT A$,A$,A$,A$,A$
```

place les tabulations sur les colonnes 1, 11, 21 et 31 et génère l'affichage suivant :

```
*         *         *         *
*
```

Si le mode 2 avait été spécifié, les tabulations auraient été définies sur les colonnes 1, 11, 21, 31, 41, 51, 61 et 71.



La fonction @ est un pointeur de variable. Elle retourne l'adresse du premier octet où est stockée, en mémoire centrale, la valeur d'une variable numérique ou celle du descripteur de chaîne dans le cas d'une variable alphanumérique.

FORMAT

x = @ variable

- **variable** peut être un nom de variable numérique ou de variable de chaîne. Il peut s'agir d'une variable indicée ou non indicée (scalaire). Elle doit être définie avant que la fonction @ ne soit appelée.
- L'adresse **x** retournée par la fonction est un nombre entier compris entre –32768 et 32767. Si ce nombre est négatif, l'adresse absolue est obtenue en ajoutant 65536 à x.

COMMENTAIRES

Pour comprendre la manière dont l'Amstrad gère le stockage et l'utilisation des variables, il est nécessaire de distinguer plusieurs éléments. Il ne suffit pas en effet que la valeur de la variable soit stockée en mémoire centrale ; encore faut-il avoir un moyen d'associer la valeur à la variable en question lorsque celle-ci est appelée dans un programme. C'est la raison pour laquelle la zone de stockage d'une variable est formée de deux groupes d'octets contigus :

- Le premier (situé vers la partie basse de la mémoire) correspond au descripteur de la variable. Celui-ci renferme les codes définissant le type (entier, réel, chaîne de caractères, tableau) et le nom de la variable. Dans le cas d'un tableau, le descripteur contient aussi des informations codées concernant l'organisation de ce tableau.
- Le second contient, sauf dans le cas d'une variable de chaîne, la valeur de la variable. Dans le cas d'une variable de chaîne, ce

second groupe d'octets contient le descripteur de la chaîne (voir ci-dessous) et non la valeur qui lui est affectée. Les notions de descripteur de chaîne et de descripteur de variable ne doivent pas être confondues.

La fonction @ retourne toujours l'adresse du premier octet de ce second groupe, c'est-à-dire l'adresse du premier octet correspondant soit à la zone de stockage de la valeur de la variable, soit à celle du descripteur de chaîne. Cette adresse sera notée &A dans les explications données ci-dessous.

Le descripteur d'une variable numérique ou d'une variable de chaîne

La structure de ce descripteur est la suivante (en allant du haut vers le bas de la mémoire) :

Octet d'adresse &A – 1

Code du type de la variable.
La valeur 1 correspond à une variable entière, 2 à une variable de chaîne et 4 à une variable réelle.

Octet d'adresse &A – 2

Code de la valeur ASCII augmentée de 128 (décimal) du dernier caractère formant le nom de la variable.

Octet d'adresse &A – 3

Code de la valeur ASCII de l'avant-dernier caractère formant le nom de la variable.

Octet d'adresse &A – n

Code de la valeur ASCII du premier caractère formant le nom de la variable.

Par exemple, le descripteur de la variable de chaîne AMSTRAD$ (qui sera un nom de variable et non une valeur) sera :

&A-8	&A-7	&A-6	&A-5	&A-4	&A-3	&A-2	&A-1	Adresse
65	77	83	84	82	65	196	2	Valeur décimale
A	M	S	T	R	A	D+128		Code ASCII

Ces correspondances peuvent être affichées à l'écran à l'aide des quelques lignes de programme suivantes :

```
10 AMSTRAD$=""
20 FOR I=8 TO 1 STEP -1
30 A=@AMSTRAD$
40 C=PEEK(A-I)
50 PRINT I,C,CHR$(C)
60 NEXT I
```

L'avant-dernière valeur affichée dans la colonne CHR$ sera naturellement incorrecte (il faut retirer 128 pour avoir le véritable numéro de code ASCII).

Le descripteur d'une variable de tableau

Les éléments figurant dans le descripteur d'une variable de tableau (par exemple A(I,J)) sont les suivants (en commençant par la partie basse de la mémoire et en allant vers les adresses élevées) :

- Le nom du tableau, la première lettre de ce nom ayant la plus petite adresse. La valeur correspondant à la dernière lettre du nom est augmentée de 128 (décimal) par rapport au numéro de code ASCII qui lui est associé.
- Le numéro (codé sur un octet) du type de la variable (1, 2 ou 4).
- La valeur du déplacement, codée sur deux octets. Le déplacement correspond au nombre d'octets séparant le champ ''valeur du déplacement'' du champ codant pour la valeur de la variable. La valeur du déplacement se calcule de la manière suivante :

  1 + 2 × (nombre d'indices dans le tableau)
  + (nombre d'octets pour stocker un élément) × (nombre d'éléments)

- Le nombre d'indices (codé sur un octet) dans le tableau (par exemple la valeur 2, dans le cas d'un tableau du type A(I,J)).

- Le nombre (codé sur deux octets) d'éléments se rapportant au premier indice (par exemple la valeur 5 dans le cas d'un tableau A(5,16)).
- Le nombre d'éléments se rapportant au deuxième indice.
- ...
- Le nombre d'éléments se rapportant au dernier indice.

Par exemple, le descripteur du tableau numérique à une dimension (ou vecteur) FIC(10) aura la structure :

&A-9	&A-8	&A-7	&A-6	&A-5	&A-4	&A-3	&A-2	&A-1	Adresse
70	73	195	4	58	0	1	11	0	Valeur déc.

Les valeurs correspondant à ce descripteur peuvent être affichées à l'écran au moyen du programme suivant :

```
10 DIM FIC(10)
20 FOR I=9 TO 1 STEP -1
30 A=@FIC(0)
40 PRINT I,PEEK(A-I)
50 NEXT I
```

Les correspondances observées ont la signification suivante :

70	Numéro de code ASCII de la lettre F.
73	Numéro de code ASCII de la lettre I.
195	Numéro de code ASCII + 128 de la lettre C.
4	Code correspondant à des valeurs réelles (tableau numérique).
58	Valeur du déplacement ($=1+2\times1+5\times11$).
0	Octet de poids fort du champ "valeur de déplacement". La valeur du déplacement étant inférieure à 128, cet octet vaut 0.
1	Nombre d'indices (un seul pour un vecteur).
11	Nombre d'éléments se rapportant au premier (et ici au seul)

indice. Il y a effectivement 11 éléments puisque la numérotation commence à 0 (éléments FIC(0) à FIC(10)).

Les valeurs des variables numériques

Les valeurs des variables entières sont stockées sur deux octets consécutifs en mémoire centrale (octets d'adresses &A et &A+1, pour un scalaire) ; les valeurs des variables numériques réelles sont stockées sur cinq octets consécutifs. Dans les deux cas, l'octet de poids faible est celui dont l'adresse est la plus basse. Les valeurs d'un tableau numérique sont stockées les unes à la suite des autres, en commençant par les valeurs correspondant aux indices 0.

La manière dont la valeur d'une variable réelle est codée sur les cinq octets qui lui sont réservés est assez lourde à détailler. Elle ne sera pas présentée ici, l'utilisateur intéressé ne devant avoir aucune peine à établir les correspondances au moyen de petits programmes semblables à ceux qui sont donnés ci-dessus. Signalons seulement que le bit 7 de l'octet d'adresse &A+3 code pour le signe du nombre (ce bit vaut 0 pour un nombre positif et 1 pour un nombre négatif).

Les descripteurs de chaînes des variables alphanumériques

Les variables de chaîne sont stockées de manière dynamique en mémoire centrale. A chaque variable de ce type est associée une suite de trois octets appelée ''descripteur de chaîne''. La valeur du premier octet du descripteur représente la longueur de la chaîne ; les deux octets suivants contiennent l'adresse où est stockée la chaîne en mémoire, l'octet de poids faible ayant l'adresse la plus basse. Comme indiqué plus haut, la fonction @ retourne l'adresse du premier octet du descripteur de chaîne d'une telle variable.

Le programme donné ci-dessous permet d'afficher à l'écran l'endroit où est stockée, en mémoire centrale, une variable de chaîne entrée au clavier (il s'agit de l'adresse de la chaîne elle-même, et non de celle de son descripteur).

```
10 INPUT "Entrez une chaîne de caractères quelconque";a$
15 A=@a$
20 B=PEEK(@+1)+(PEEK(@+2)*256)
30 PRINT "Adresse de la chaîne en mémoire :";B
40 END
```

INDEX

LA BIBLIOTHÈQUE SYBEX

OUVRAGES GÉNÉRAUX

VOTRE PREMIER ORDINATEUR *par* RODNAY ZAKS,
296 pages, Réf. 394

VOTRE ORDINATEUR ET VOUS *par* RODNAY ZAKS,
296 pages, Réf. 271

DU COMPOSANT AU SYSTÈME : une introduction aux microprocesseurs *par* RODNAY ZAKS,
636 pages, Réf. 0040

TECHNIQUES D'INTERFACE aux microprocesseurs *par* AUSTIN LESEA ET RODNAY ZAKS,
450 pages, Réf. 0039

LEXIQUE INTERNATIONAL MICRO-ORDINATEURS, avec dictionnaire abrégé en 10 langues
192 pages, Réf. 234

GUIDE DES MICRO-ORDINATEURS A MOINS 3 000 F *par* JOËL PONCET,
144 pages, Réf. 322

LEXIQUE MICRO-INFORMATIQUE *par* PIERRE LE BEUX,
140 pages, Réf. 369

LA SOLUTION RS-232 *par* JOE CAMPBELL,
208 pages, Réf. 0052

MINITEL ET MICRO-ORDINATEUR *par* PIERRICK BOURGAULT,
198 pages, Réf. 0119

ROBOTS - CONSTRUCTION, PROGRAMMATION *par* FERNAND ESTEVES,
400 pages, Réf. 0130

ALGORITHMES *par* PIERRE BEAUFILS ET WOLFRAM LUTHER,
296 pages, Réf. 0149

BASIC

VOTRE PREMIER PROGRAMME BASIC *par* RODNAY ZAKS,
208 pages, Réf. 263

INTRODUCTION AU BASIC *par* PIERRE LE BEUX,
336 pages, Réf. 0035

LE BASIC PAR LA PRATIQUE : 60 exercices *par* JEAN-PIERRE LAMOITIER,
252 pages, Réf. 0095

LE BASIC POUR L'ENTREPRISE *par* XUAN TUNG BUI,
204 pages, Réf. 253

PROGRAMMES EN BASIC, Mathématiques, Statistiques, Informatique *par* ALAN R. MILLER,
318 pages, Réf. 259

BASIC, PROGRAMMATION STRUCTURÉE *par* RICHARD MATEOSIAN,
352 pages, Réf. 0129

JEUX D'ORDINATEUR EN BASIC *par* DAVID H. AHL,
192 pages, Réf. 246

NOUVEAUX JEUX D'ORDINATEUR EN BASIC *par* DAVID H. AHL,
204 pages, Réf. 247

FICHIERS EN BASIC *par* ALAN SIMPSON,
256 pages, Réf. 0102

TECHNIQUES DE PROGRAMMATION EN BASIC *par* S. CROSMARIE, M. PERRON ET D. PHILIPPINE
152 pages, Réf. 0124

PASCAL

INTRODUCTION AU PASCAL *par* PIERRE LE BEUX,
496 pages, Réf. 0030

LE PASCAL PAR LA PRATIQUE *par* PIERRE LE BEUX ET HENRI TAVERNIER,
562 pages, Réf. 361

LE GUIDE DU PASCAL *par* JACQUES TIBERGHIEN,
504 pages, Réf. 423

PROGRAMMES EN PASCAL pour Scientifiques et Ingénieurs *par* ALAN R. MILLER,
392 pages, Réf. 240

AUTRES LANGAGES

INTRODUCTION A ADA *par* PIERRE LE BEUX,
366 pages, Réf. 360

INTRODUCTION A C *par* BRUCE HUNTER,
312 pages, Réf. 0092

MICRO-ORDINATEURS

ALICE

JEUX EN BASIC POUR ALICE *par* PIERRE MONSAUT,
96 pages, Réf. 320

ALICE et ALICE 90, PREMIERS PROGRAMMES *par* RODNAY ZAKS,
248 pages, Réf. 376

ALICE, 56 PROGRAMMES *par* STANLEY R. TROST,
160 pages, Réf. 401

ALICE, GUIDE DE L'UTILISATEUR *par* NORBERT RIMOUX,
208 pages, Réf. 378

ALICE, PROGRAMMATION EN ASSEMBLEUR *par* GEORGES FAGOT-BARRALY,
192 pages, Réf. 420

AMSTRAD

AMSTRAD, PREMIERS PROGRAMMES *par* RODNAY ZAKS,
248 pages, Réf. 0105

AMSTRAD, 56 PROGRAMMES *par* STANLEY R. TROST,
160 pages, Réf. 0107

AMSTRAD, JEUX D'ACTION *par* PIERRE MONSAUT,
96 pages, Réf. 0108

AMSTRAD, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
208 pages, Réf. 0136

AMSTRAD EXPLORÉ *par John Braga,*
192 pages, Réf. 0135

AMSTRAD, GUIDE DU GRAPHISME *par James Wynford,*
208 pages, Réf. 0141

AMSTRAD CP/M 2.2 *par Anatole D'Hardencourt,*
248 pages, Réf. 0156

AMSTRAD ASTROLOGIE/NUMEROLOGIE/BIORYTHMES
par Pierrick Bourgault,
160 pages, Réf. 0167

AMSTRAD MULTIPLAN *de Microsoft,*
496 pages, Réf. 1111

AMSTRAD, CRÉER DE NOUVELLES INSTRUCTIONS
par Jean-Claude Despoine,
144 pages, Réf. 0176

AMSTRAD ASTROCALC
par Gérard Blanc et Philippe Destrebecq,
168 pages, Réf. 0162

APPLE / MACINTOSH

PROGRAMMEZ EN BASIC SUR APPLE II,
Tomes 1 et 2 *par Léopold Laurent,*
208 pages, Réf. 333 et 380

APPLE II 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 283

JEUX EN PASCAL SUR APPLE
par Douglas Hergert et Joseph T. Kalash,
372 pages, Réf. 241

GUIDE DU BASIC APPLE II *par Douglas Hergert,*
272 pages, Réf. 0006

APPLE II, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 373

MACINTOSH, GUIDE DE L'UTILISATEUR
par Joseph Caggiano,
208 pages, Réf. 396

APPLE IIC, GUIDE DE L'UTILISATEUR
par Thomas Blackadar,
160 pages, Réf. 0089

MULTIPLAN SUR MACINTOSH
par Goulven Habasque,
240 pages, Réf. 0099

INTRODUCTION A MAC PASCAL *par Pierre Le Beux,*
416 pages, Réf. 0145

MACINTOSH POUR LA PRESSE, L'EDITION ET
LA PUBLICITE *par Bernard Le Du,*
160 pages, Réf. 0173

ATARI

JEUX EN BASIC SUR ATARI *par Paul Bunn,*
96 pages, Réf. 282

ATARI, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 387

ATARI, GUIDE DE L'UTILISATEUR *par Thomas Blackadar,*
192 pages, Réf. 354

ATMOS

JEUX EN BASIC SUR ATMOS *par Pierre Monsaut,*
96 pages, Réf. 346

ATMOS, 56 PROGRAMMES *par Stanley R. Trost,*
180 pages, Réf. 372

COMMODORE 64

JEUX EN BASIC SUR COMMODORE 64
par Pierre Monsaut,
96 pages, Réf. 0017

COMMODORE 64, PREMIERS PROGRAMMES
par Rodnay Zaks,
248 pages, Réf. 342

GUIDE DU BASIC VIC 20, COMMODORE 64
par Douglas Hergert,
240 pages, Réf. 312

COMMODORE 64, GUIDE DE L'UTILISATEUR
par J. Kascmer,
144 pages, Réf. 314

COMMODORE 64, 66 PROGRAMMES
par Stanley R. Trost,
192 pages, Réf. 319

COMMODORE 64, GUIDE DU GRAPHISME
par Charles Platt,
372 pages, Réf. 0053

COMMODORE 64, JEUX D'ACTION *par Eric Ravis,*
96 pages, Réf. 403

COMMODORE 64, 1ers CONTACTS
par Marty Dejonghe et Caroline Earhart,
208 pages, Réf. 390

COMMODORE 64, BASIC APPROFONDI
par Gary Lippman,
216 pages, Réf. 0100

DRAGON

JEUX EN BASIC SUR DRAGON *par Pierre Monsaut,*
96 pages, Réf. 324

EXL 100

EXL 100, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0126

GOUPIL

PROGRAMMEZ VOS JEUX SUR GOUPIL
par François Abella,
208 pages, Réf. 264

HECTOR

HECTOR JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 388

IBM

IBM PC EXERCICES EN BASIC *par Jean-Pierre Lamoitier,*
256 pages, Réf. 338

IBM PC GUIDE DE L'UTILISATEUR
par Joan Lasselle et Carol Ramsey,
160 pages, Réf. 301

IBM PC 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 359

GRAPHIQUES SUR IBM PC *par Nelson Ford,*
320 pages, Réf. 357

GUIDE DE PC DOS *par Richard A. King,*
240 pages, Réf. 0013

LASER

LASER JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 371

MO 5

MO 5 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0067

MO 5, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 370

MO 5, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 375

MO 5, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
192 pages, Réf. 384

MO 5, DYNAMIQUE CINÉMATIQUE, MÉTHODE POUR LA PROGRAMMATION DES JEUX *par Daniel Lebigre,*
272 pages, Réf. 0118

MO 5, STATIQUE, DYNAMIQUE, ELECTRONIQUE, PROGRAMMES DE PHYSIQUE EN BASIC
par Beaufils, Lamarche et Muggianu,
240 pages, Réf. 0148

MO 5, PROGRAMMES D'ELECTRONIQUE EN BASIC
par Beaufils, Delusurieux, Do, Romanacce,
312 pages, Réf. 0143

MO 5, OPTIQUE, THERMODYNAMIQUE, CHIMIE
par P. Beaufils, M. Lamarche, Y. Muggianu,
224 pages, Réf. 0161

MSX

MSX, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 411

MSX, INITIATION AU BASIC *par Rodnay Zaks,*
248 pages, Réf. 410

MSX, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0109

MSX, GUIDE DU GRAPHISME *par Mike Shaw,*
192 pages, Réf. 0132

MSX, PROGRAMMES EN LANGAGE MACHINE
par Steeve Webb,
112 pages, Réf. 0153

MSX, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
216 pages, Réf. 0144

MSX, GUIDE DU BASIC *par Michel Laurent,*
264 pages, Réf. 0155

MSX, JEUX EN ASSEMBLEUR *par Eric Ravis*
112 pages, Réf. 0170

MSX, ROUTINES GRAPHIQUES EN ASSEMBLEUR
par Steeve Webb
88 pages, Réf. 0154

MSX, TECHNIQUES DE PROGRAMMATION DES JEUX EN ASSEMBLEUR
par Georges Fagot-Barraly,
176 pages, Réf. 0178

MSX ASTROLOGIE/NUMEROLOGIE/BIORYTHMES
par Pierrick Bourgault,
157 pages, Réf. 0168

ORIC

JEUX EN BASIC SUR ORIC *par Peter Shaw,*
96 pages, Réf. 278

ORIC PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 344

SHARP

DÉCOUVREZ LE SHARP PC-1500 ET LE TRS-80 PC-2
par Michel Lhoir,
2 tomes, Réf. 261-262

SPECTRAVIDEO

SPECTRAVIDEO, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 377

SPECTRUM

PROGRAMMEZ EN BASIC SUR SPECTRUM
par S.M. Gee,
208 pages, Réf. 252

JEUX EN BASIC SUR SPECTRUM *par Peter Shaw,*
96 pages, Réf. 276

SPECTRUM, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 381

SPECTRUM JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 368

TI 99/4

PROGRAMMEZ VOS JEUX SUR TI 99/4
par François Abella,
160 pages, Réf. 303

TO 7

JEUX EN BASIC SUR TO 7 *par Pierre Monsaut,*
96 pages, Réf. 0026

TO 7, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 328

TO 7, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
192 pages, Réf. 350

JEUX SUR TO 7 et MO 5 *par Georges Fagot-Barraly,*
168 pages, Réf. 0134

GESTION DE FICHIERS SUR TO 7 ET MO 5
par Jean-Pierre Lhoir,
136 pages, Réf. 0127

TO 7, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 374

TO 7 / MO 5, GUIDE DU BASIC
par Jean-Louis Greco et Michel Laurent,
288 pages, Réf. 0158

TO 7 / MO 5, GUIDE DU GRAPHISME
par Michel Lamarche et Yves Muggianu,
240 pages, Réf. 0172

TO 7 / MO 5 ASTROLOGIE/NUMEROLOGIE/BIORYTHMES
par Pierrick Bourgault,
160 pages, Réf. 0169

TRS-80

PROGRAMMEZ EN BASIC SUR TRS-80
par Léopold Laurent,
2 tomes, Réf. 366-251

JEUX EN BASIC SUR TRS-80 MC-10 *par Pierre Monsaut,*
96 pages, Réf. 323

JEUX EN BASIC SUR TRS-80 *par Chris Palmer,*
96 pages, Réf. 302

JEUX EN BASIC SUR TRS-80 COULEUR
par Pierre Monsaut,
96 pages, Réf. 325

TRS-80 MODÈLE 100, GUIDE DE L'UTILISATEUR
par Orson Kellog,
112 pages, Réf. 300

TRS-80 COULEUR, PREMIERS PROGRAMMES
par Rodnay Zaks,
248 pages, Réf. 414

TRS-80 COULEUR, 56 PROGRAMMES
par Stanley R. Trost,
160 pages, Réf. 413

VIC 20

PROGRAMMEZ EN BASIC SUR VIC 20
par G. O. Hamann,
2 tomes, Réf. 329-337

JEUX EN BASIC SUR VIC 20 *par Alastair Gourlay,*
96 pages, Réf. 277

VIC 20, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 341

VIC 20 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 345

VG 5000

VG 5000, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 422

VG 5000, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0128

ZX 81

ZX 81 GUIDE DE L'UTILISATEUR *par Douglas Hergert,*
208 pages, Réf. 351

ZX 81 56 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 304

GUIDE DU BASIC ZX 81 *par Douglas Hergert,*
204 pages, Réf. 285

JEUX EN BASIC SUR ZX 81 *par Mark Charlton,*
96 pages, Réf. 275

ZX 81 PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 343

MICROPROCESSEURS

PROGRAMMATION DU Z80 *par Rodnay Zaks,*
618 pages, Réf. 0058

APPLICATIONS DU Z80 *par James W. Coffron,*
304 pages, Réf. 0181

PROGRAMMATION DU 6502 *par Rodnay Zaks,*
376 pages, Réf. 0031, 2ème édition

APPLICATIONS DU 6502 *par Rodnay Zaks,*
288 pages, Réf. 332

PROGRAMMATION DU 6800
par Daniel-Jean David et Rodnay Zaks,
374 pages, Réf. 327

PROGRAMMATION DU 6809
par Rodnay Zaks et William Labiak,
392 pages, Réf. 0139

PROGRAMMATION DU 8086/8088
par James W. Coffron,
304 pages, Réf. 0016

MISE EN OEUVRE DU 68000 *par C. Vieillefond,*
352 pages, Réf. 0133

ASSEMBLEUR DU 8086/8088 *par François Retoreau,*
616 pages, Réf. 0093

SYSTÈMES D'EXPLOITATION

GUIDE DU CP/M AVEC MP/M *par Rodnay Zaks,*
354 pages, Réf. 336

CP/M APPROFONDI *par Alan R. Miller,*
380 pages, Réf. 334

INTRODUCTION AU p-SYSTEM UCSD
par Charles W. Grant et Jon Butah,
308 pages, Réf. 365

GUIDE DE MS-DOS *par Richard A. King,*
360 pages, Réf. 0117

INTRODUCTION A UNIX *par John D. Halamka,*
240 pages, Réf. 0098

GUIDE DE PRODOS
par Pierre Beaufils et Wolfram Luther,
248 pages, Réf. 0146

APPLICATIONS ET LOGICIELS

INTRODUCTION AU TRAITEMENT DE TEXTE
par Hal Glatzer,
228 pages, Réf. 243

INTRODUCTION A WORDSTAR *par Arthur Naiman,*
200 pages, Réf. 0062

WORDSTAR APPLICATIONS *par* *Julie Anne Arca,*
320 pages, Réf. 0005

VISICALC APPLICATIONS *par* *Stanley R. Trost,*
304 pages, Réf. 258

VISICALC POUR L'ENTREPRISE *par* *Dominique Helle,*
304 pages, Réf. 309

INTRODUCTION A dBASE II *par* *Alan Simpson,*
280 pages, Réf. 0064

DE VISICALC A VISI ON *par* *Jacques Bourdeu,*
256 pages, Réf. 321

MULTIPLAN POUR L'ENTREPRISE
par D. Helle et G. Boussand,
304 pages, Réf. 0079

dBASE II APPLICATIONS *par* *Christophe Stehly,*
248 pages, Réf. 416

INTRODUCTION A LOTUS 1-2-3
par Chris Gilbert et Laurie Williams,
272 pages, Réf. 0106

INTRODUCTION A dBASE III *par* *Alan Simpson,*
272 pages, Réf. 0131

LOTUS 1-2-3 POUR L'ENTREPRISE
par Dominique Helle et Guy Boussand,
256 pages, Réf. 0147

LOTUS 1-2-3 PROGRAMMATION DES MACRO-COMMANDES *par* *Goulven Habasque,*
144 pages, Réf. 0150 F

LOGISTAT, ANALYSE STATISTIQUE DES DONNÉES
par Fredj Tekaia et Michele Bidel,
352 pages, Réf. 0115

ALGORITHMES *par* *P. Beaufils, et W. Luther,*
296 pages, Réf. 0149

POUR UN CATALOGUE COMPLET DE NOS PUBLICATIONS

FRANCE
6-8, Impasse du Curé
75881 PARIS CEDEX 18
Tél. : (1) 42.03.95.95
Télex : 211801

U.S.A.
2344 Sixth Street
Berkeley, CA 94710
Tel. : (415) 848.8233
Telex : 336311

ALLEMAGNE
Vogelsanger. WEG 111
4000 Düsseldorf 30
Postfach N° 30.09.61
Tel. : (0211) 61 80 2-0
Telex : 08588163



Paris • Berkeley • Düsseldorf

Achevé d'imprimer le 10 février 1986 sur les presses de l'Imprimerie «La Source d'Or»
63200 Marsat - Dépôt légal : 1er trimestre 1986 - Imprimeur n° 1935

Ce guide est un dictionnaire complet du BASIC Amstrad disponible sur les modèles CPC 464, CPC 664 et CPC 6128. Chaque instruction, commande ou fonction est présentée, commentée et illustrée par des exemples de programmes. L'étude de ces exemples permettra au lecteur de mieux exploiter les possibilités de son ordinateur. Certains de ces programmes pourront même être utilisés directement ou être intégrés à des programmes plus importants.



LES AUTEURS

Michel Laurent est chercheur au CNRS. Jean Louis Greco est ingénieur informaticien au CEA. Tous deux ont développé de nombreux programmes d'applications scientifiques.




SYBEX
AMSTRAD GUIDE DU BASIC ET DE L'AMSDOS



Document numérisé avec amour par

AMSTRAD CPC

MÉMOIRE ÉCRITE



https://acpc.me/